Introduction to Information Security

Lecture 6: Public Key Cryptography

2007. 6.

Prof. Byoungcheon Lee
sultan (at) joongbu . ac . kr

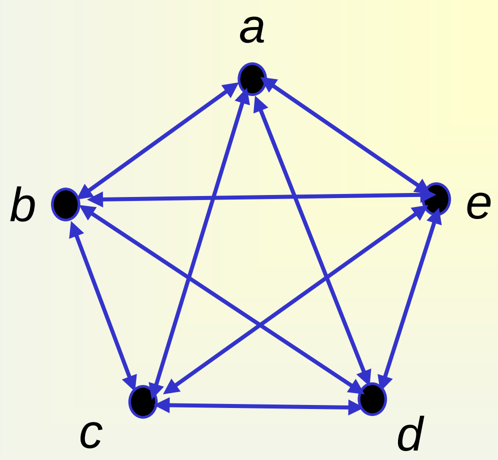
Information and Communications University

Contents

1. Introduction to PKC

Key Distribution Problem of Symmetric Key Crypto

- **In symmetric key cryptosystems**
  - Over complete graph with *n* nodes, ${}_{n}C_{2} = n(n-1)/2$ pairs secret keys are required.
  - (Example) n=100, 99 x 50 = 4,950 keys are required
  - Problem: Managing large number of keys and keeping them in a secure manner is difficult

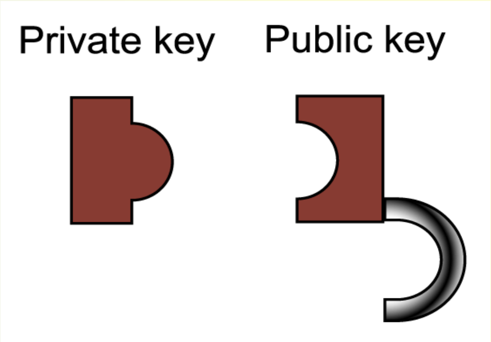


Secret keys are required between

(a,b), (a,c), (a,d), (a,e), (b,c), (b,d), (b,e), (c,d), (c,e), (d,e)

Public Key Cryptography - Concept

Using a pair of keys which have special mathematical relation.
Each user needs to keep securely only his private key.
All public keys of users are published.



In Encryption

Anyone can lock (using the public key)
Only the receiver can unlock (using the private key)

In Digital Signature

Only the signer can sign (using the private key)
Anyone can verify (using the public key)

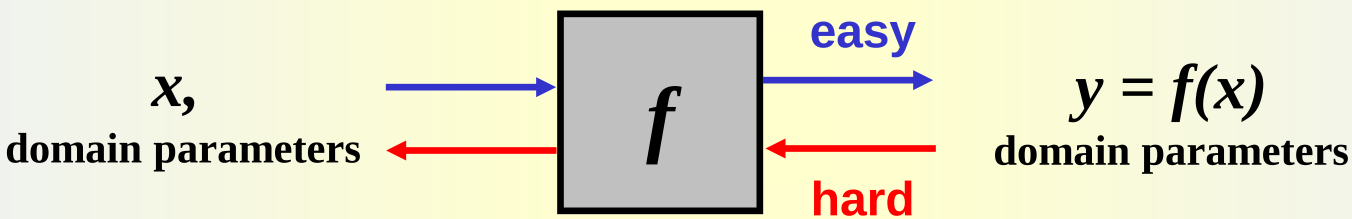
Symmetric key vs. Asymmetric Key Crypto

O : merit
X : demerit

	Symmetric		Asymmetric
Key relation	Enc. key = Dec. key		Enc. Key ≠ Dec. key
Enc. Key	Secret		Public, {private}
Dec. key	Secret		Private, {public}
Algorithm	Secret	Public	Public
Example	SKIPJACK	AES	RSA
Key Distribution	Required (X)		Not required (O)
Number of keys	Many (X)		Small (O)
E/D Speed	Fast(O)		Slow(X)

Public Key Cryptography - Concept

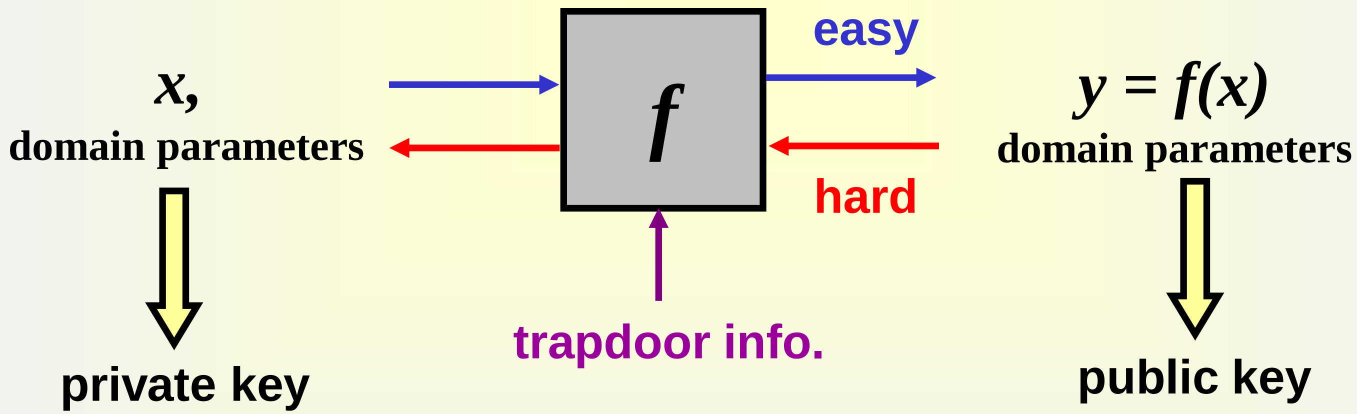
- **One-way functions**
  - Given *x*, easy to compute $y=f(x)$.
  - Difficult to compute $x=f^{-1}(y)$ for given *y*.



Ex) $f(x)= 7x^{21} + 3x^{3} + 13x^{2}+1 \bmod (2^{15}-1)$

Public Key Cryptography - Concept

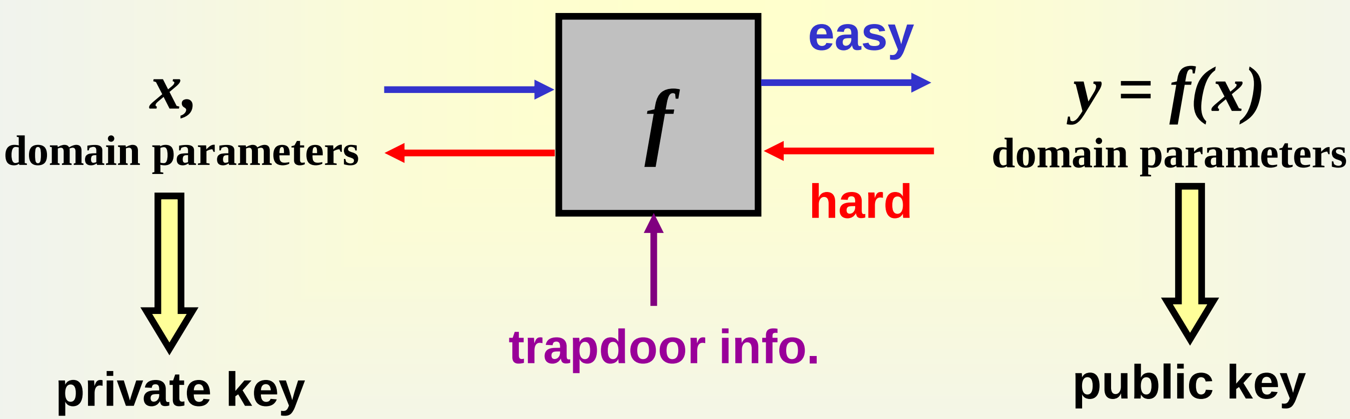
- **Trapdoor one-way functions**
  - Given *x*, easy to compute *f(x)*
  - Given *y*, difficult to compute $f^{-1}(y)$ in general
  - Easy to compute $f^{-1}(y)$ for given *y* to only who knows certain information (which we call trapdoor information)



But, easy if trapdoor info. is given.

Public Key Cryptography - Concept

- Concept
  - invented by Diffie and Hellman in 1976, "*New directions in Cryptography*", IEEE Tr. on IT. ,Vol. 22, pp. 644-654, Nov., 1976.
  - Overcome the problem of secret key sharing in symmetric cryptosystems
  - Two keys used: public key & private key
  - Also known as two-key cryptography or asymmetric cryptography
  - Based on (trapdoor) one-way function



But, easy if trapdoor info. is given.

Public Key Cryptography

- Keys
  - ✓ A pair of (Public Key, Private Key) for each user
  - ✓ Public keys must be publicly & reliably available
- Encryption schemes
  - ✓ Encrypt with peer's Public Key; Decrypt with its own Private Key
  - ✓ RSA, ElGamal
- Digital signature schemes
  - ✓ Sign with its own Private Key; verify with peer's Public Key
  - ✓ RSA, DSA, KCDSA, ECDSA, EC-KCDSA …
- Key exchange schemes
  - ✓ Key transport or key agreement for secret-key crypto.
  - ✓ RSA; DH(Diffie-Hellman), ECDH

- All problems clear?
  - ✓ New Problem : How to get the right peer's Public Key?
  - ✓ Public key infrastructure (PKI) required
  - ✓ Certificate is used to authenticate public key

Public Key Cryptosystems

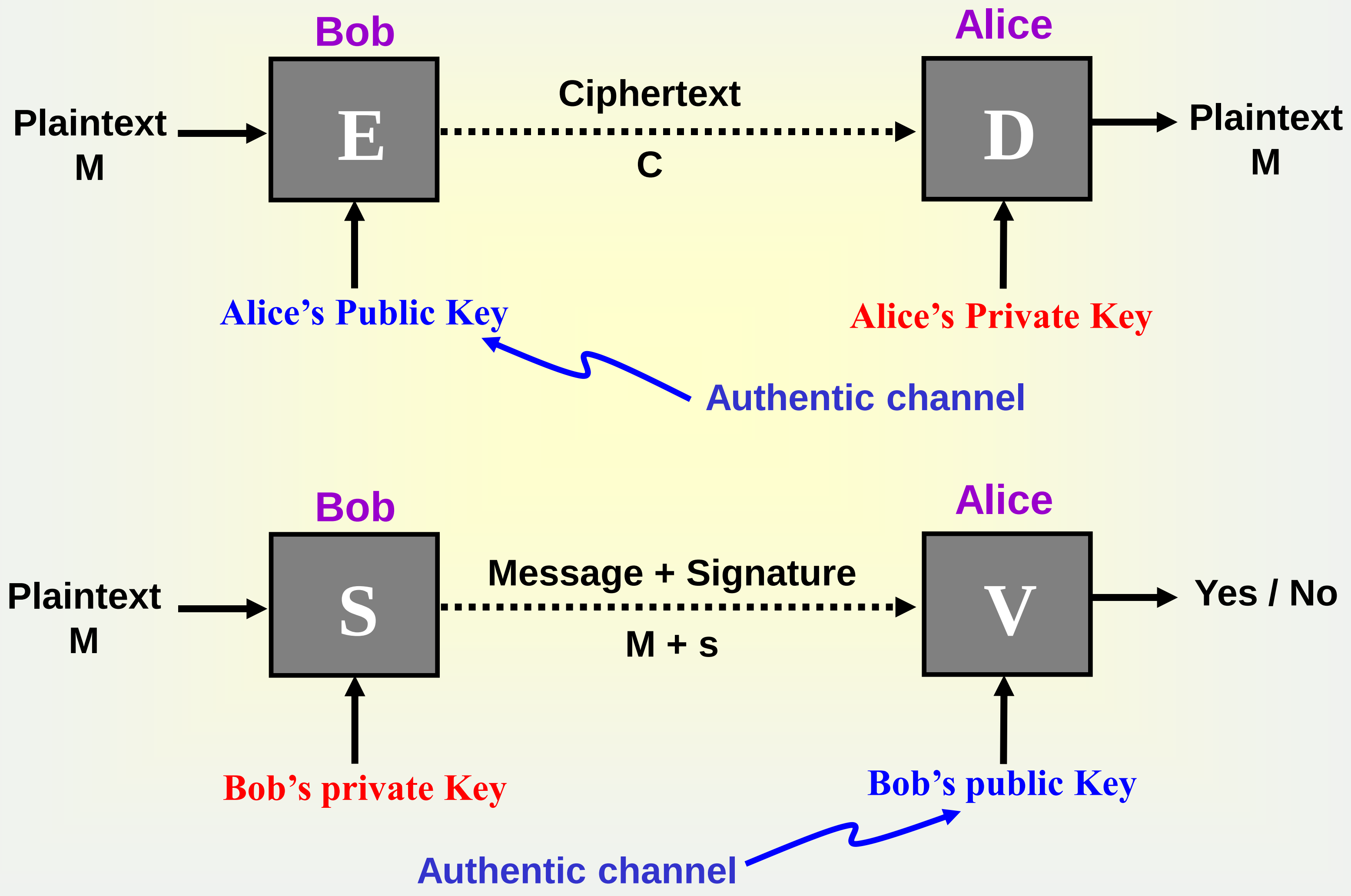
- Public key cryptography is based on hard problems.

- Encryption schemes
  - RSA: based on IFP
  - ElGamal: based on DLP

- Signature schemes
  - Signature schemes with message recovery: RSA
  - Signature with appendix: ElGamal, DSA, KCDSA

- Key exchange schemes
  - Key transport: a trusted entity TA generates and distributes key
  - Key agreement: Diffie-Hellman key agreement. Both entity take part in the key agreement process to have an agreed key

Public Key Encryption vs. Digital Signature



Public Key Cryptosystems – History

- RSA scheme (1978)
  - *R.L.Rivest, A.Shamir, L.Adleman, "A Method for Obtaining Digital Signatures and Public Key Cryptosystems",CACM, Vol.21, No.2, pp.120-126,Feb,1978*
- McEliece scheme (1978)
- Rabin scheme (1979)
- Knapsack scheme (1979-): Merkle-Hellman, Chor-Rivest
- ElGamal scheme (1985)
- Elliptic Curve Cryptosystem (1985): Koblitz, Miller
- Non-Abelian group Cryptography (2000): Braid group

2. Hard Problems

IFP (Integer Factorization Problem)

DLP (Discrete Logarithm Problem)

Integer Factorization Problem (IFP)

- **Problem: Given a composite number n, find its prime factors**

$$\text{Primes } p, q \;\; \underset{\text{hard}}{\overset{\text{easy}}{\rightleftarrows}} \;\; n = pq$$

- **Application: Used to construct RSA-type public key cryptosystems**

- **Algorithms to solve IFP (probabilistic sub-exponential algorithms)**
  - **Quadratic sieve**
  - **General Number Field Sieve**

Quadratic Sieve

- **Factor n (=pq) using the quadratic sieve algorithm**

- **Basic principle:**
  Let n be an integer and suppose there exist integers x and y with $x^2 = y^2 \pmod{n}$, but $x \neq \pm y \pmod{n}$. Then gcd(x-y,n) gives a nontrivial factor of n.

- **Example**
  Consider n=77
  72=-5 mod 77, 45=-32 mod 77
  72*45 = (-5)*(-32) mod 77
  $2^3 \cdot 3^4 \cdot 5 = 2^5 \cdot 5 \bmod 77$
  $9^2 = 2^2 \bmod 77$
  gcd(9-2,77)=7, gcd(9+2,77)=11
  77=11*7 Factorization

Quadratic Sieve

- **Example: factor n=3837523.**
  (textbook p. 183)

Observe

$$9398^2 = 5^5 \times 19 \pmod{3837523}$$
$$19095^2 = 2^2 \times 5 \times 11 \times 13 \times 19 \pmod{3837523}$$
$$1964^2 = 3^2 \times 13^3 \pmod{3837523}$$
$$17078^2 = 2^6 \times 3^2 \times 11 \pmod{3837523}$$

Then we have

$$(9398 \times 19095 \times 1964 \times 17078)^2 = (2^4 x 3^2 x 5^3 x 11 x 13^2 x 19)^2$$
$$2230387^2 = 2586705^2 \pmod{3837523}$$
$$\gcd(2230387-2586705, 3837523)=1093$$
$$3837523 / 1093 = 3511$$

3837523 = 1093 x 3511 ← succeed !

Quadratic Sieve

- Quadratic Sieve algorithm : find factors of integer n
  1. Initialization: a sequence of quadratic residues $Q(x)=(m+x)^2-n$ is generated for small values of x where $m=\lfloor sqrt(n) \rfloor$.
  2. Forming the factor base: the base consists of small primes. $FB=\{-1,2,p_1,p_2,\ldots,p_{t-1}\}$
  3. Sieving: the quadratic residues Q(x) are factored using the factor base till t full factorizations of Q(x) have been found.
  4. Forming and solving the matrix: Find a linear combination of Q(x)'s which gives the quadratic congruence. The congruence gives a nontrivial factor of n with the probability ½.

http://www.answers.com/topic/quadratic-sieve?cat=technology

- Exercise 1: Find factors of n=4841 using the quadratic sieve algorithm

General Number Field Sieve (GNFS)

- **GNFS (general number field sieve) is the most efficient algorithm known for factoring integers larger than 100 digits.**

- **Asymptotic running time: sub-exponential**

$$L_n[\frac{1}{3},1.526] = O\left(e^{(1.526+o(1))(\ln n)^{1/3}(\ln\ln n)^{2/3}}\right)$$

Complexity of algorithm

$$L_n[\alpha, c] = O(e^{c(\ln n)^{\alpha}(\ln\ln n)^{1-\alpha}})$$

- If α=0, polynomial time algorithm
- If α>=1, exponential time algorithm
- If 0<α<1, sub-exponential time algorithm

ln *n* : number of bits of *n*

RSA Challenge

Digits	Year	MIPS-year	Algorithm
RSA-100	‘91.4.	7	Q.S.
RSA-110	‘92.4.	75	Q.S.
RSA-120	‘93.6.	830	Q.S.
RSA-129	‘94.4.(AC94)	5,000	Q.S.
RSA-130	‘96.4.(AC96)	?	NFS
RSA-140	‘99.2 (AC99)	?	NFS
RSA-155	’99.8	8,000	GNFS
RSA-160	’03.1		Lattice Sieving + HW
RSA-174	’03.12		Lattice Sieving + HW
RSA-200	‘05.5		Lattice Sieving + HW

- MIPS : 1 Million Instruction Per Second for 1 yr = 3.1 x 10^{13} instruction
- http://www.rsasecurity.com./rsalabs, expectation : 768-bit by 2010, 1024-bit by 2018

RSA Challenge Solution

RSA-160

Date: Tue, 1 Apr 2003 14:05:10 +0200
From: Jens Franke
Subject: RSA-160

We have factored RSA160 by gnfs. The prime factors are:
p=45427892858481394071686190649738831\ 656137145778469793250959984709250004157335359
q=47388090603832016196633832303788951\ 973268922921040957944741354648812028493909367

http://www.loria.fr/~zimmerma/records/rsa160

RSA-200

Date: Mon, 9 May 2005 18:05:10 +0200 (CEST)
From: Thorsten Kleinjung
Subject: rsa200

We have factored RSA200 by GNFS. The factors are
p=3532461934402770121272604978198464368671197400197 6\ 250236493034687761212536794232000585479565280883 49
and
q=7925869954478333033347085841480059687737975857364 2\ 199607343303414557678728181521353814093047401854 67

http://www.loria.fr/~zimmerma/records/rsa200

Discrete Logarithm Problem (DLP)

- **Problem:**
  Given g, y, and prime p, find an integer x, if any, such that $y = g^x \bmod p$ ($x=\log_g y$)

Given g, x, p	$\xrightarrow{\text{easy}}$	$y = g^x \bmod p$
$x = \log_g y$	$\xleftarrow[\text{hard}]{}$	Given g, y, p

- **Application: Used to construct Diffie-Hellman & ElGamal-type public key systems: DH, DSA, KCDSA …**

- **Algorithms to solve DLP:**
  - **Shank's Baby Step Giant Step**
  - **Index calculus**

Shank's Baby Step, Giant Step algorithm

- **Problem: find an integer x, if any, such that $y = g^x \bmod p$ ($x = \log_g y$)**
- **Algorithm**

1. Choose an integer $N = \left\lfloor \sqrt{p-1} \right\rfloor$

2. Computes $g^j \bmod p, \text{ for } 0 \le j < N$ **Baby Step**

3. Computes $yg^{-Nk} \bmod p, \text{ for } 0 \le k < N$ **Giant Step**

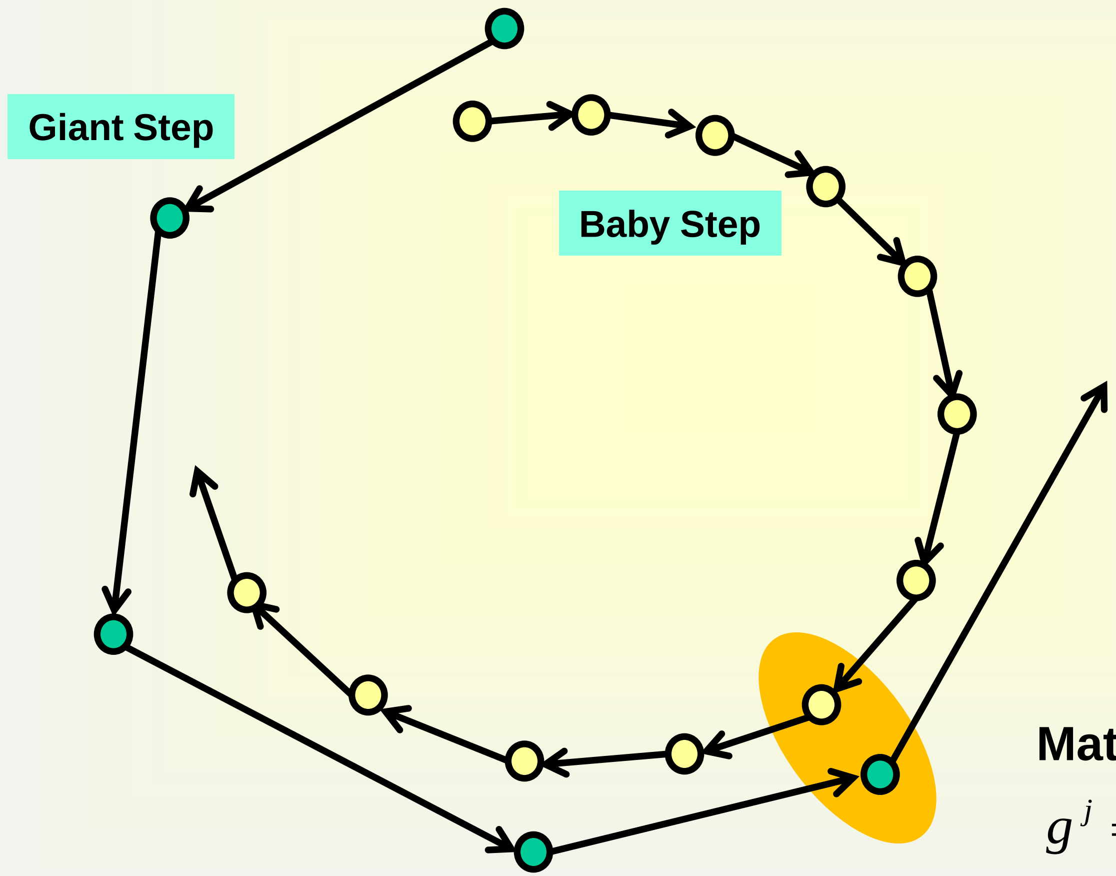
4. Look for a match between the two lists. If a match is found,

$$g^j = yg^{-Nk} \bmod p$$

Then $y = g^x = g^{j+Nk}$

We solve the DLP. $x = j + Nk$

Shank's Baby Step, Giant Step algorithm



Match found!

$$g^{j} = yg^{-Nk} \bmod p$$

Index Calculus

- **Problem: find an integer x, if any, such that $y = g^x \bmod p$ ($x=\log_g y$)**

- **Algorithm**

1. Choose a factor base $S=\{p_1,p_2,\ldots p_m\}$ which are primes less than a bound B.

2. Collect linear relations

1. **Select a random integer *k* and compute g^k mod p**
2. **Try to write g^k as a product of primes in *S***

$$g^k = \prod_i p_i^{a_i} \bmod p, \quad \text{then} \ \ k = \sum_i a_i \log_g p_i \bmod p-1$$

3. Find the logarithms of elements in S solving the linear relations

4. Find x

For a random r, compute yg^r mod p and try to write it as a product of primes in S.

$$yg^r = \prod_i p_i^{b_i} \bmod p, \quad \text{then} \ \ x = -r + \sum_i b_i \log_g p_i \bmod p-1$$

Index Calculus

- **Example: Let p=131, g=2, y=37. Find $x=\log_2 37 \bmod 131$**

- **Solution**

  Let B=10, S={2,3,5,7}

$2^1 = 2 \bmod 131$
$2^8 = 5^3 \bmod 131$
$2^{12} = 5 * 7 \bmod 131$
$2^{14} = 3^2 \bmod 131$
$2^{34} = 3 * 5^2 \bmod 131$

⇨

$1 = \log_2 2 \bmod 130$
$8 = 3*\log_2 5 \bmod 130$
$12= \log_2 5 + \log_2 7 \bmod 130$
$14 = 2*\log_2 3 \bmod 130$
$34 = \log_2 3 + 2*\log_2 5 \bmod 130$

⇨

$\log_2 2 = 1$
$\log_2 5 = 46$
$\log_2 7 = 96$
$\log_2 3 = 72$

$37 * 2^{43} = 3 * 5 * 7 \bmod 131$
$\mathrm{Log}_2 37 = -43 + \log_2 3 + \log_2 5 + \log_2 7 \bmod 130 = 41$

Solution : $2^{41} \bmod 131 = 37$

- **Exercise 2: Let p=809. Find $\log_3 525 \bmod 809$.**

Discrete Logarithm Problem (DLP)

- **Complexity of best known algorithm for solving DLP:**

$$L_p[\frac{1}{3}, 1.923] = O\left(e^{(1.923+o(1))(\ln p)^{1/3}(\ln\ln p)^{2/3}}\right)$$

- **Complexities of solving IFP and DLP are similar**

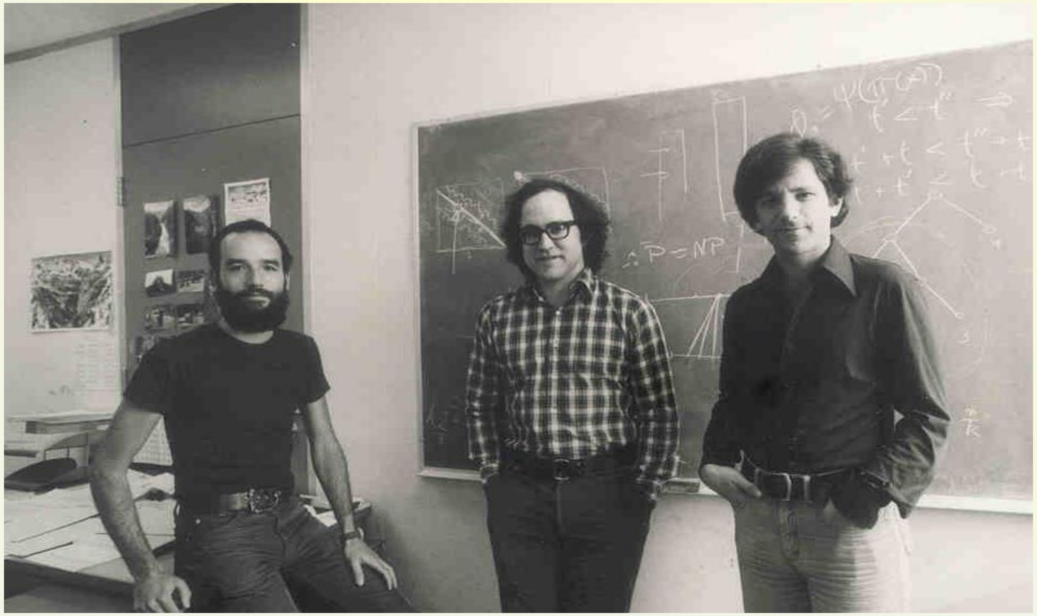
3. Public Key Encryption

RSA

ElGamal

RSA Public Key Systems

- RSA is the first public key cryptosystem
- Proposed in 1977 by Ron Rivest, Adi Shamir and Leonard Adleman at MIT
- It is believed to be secure and still widely used



Shamir *Rivest* *Adleman*

RSA Public Key Systems

- **Key generation**
  - Choose two large (512 bits or more) primes p & q
  - Compute modulus $n = pq$, and $\phi(n) = (p-1)(q-1)$
  - Pick an integer e relatively prime to $\phi(n)$, $\gcd(e, \phi(n))=1$
  - Compute d such that $ed = 1 \bmod \phi(n)$
  - Public key (n, e) : publish
  - Private key d : keep secret (may discard p & q)

- **Special Property**
  - $(m^e \bmod n)^d \bmod n = (m^d \bmod n)^e \bmod n$ for $0 < m < n$

- **Encryption / Decryption**
  - E: $c = m^e \bmod n$ for $0 < m < n$
  - D: $m = c^d \bmod n$
  - Proof) $C^d = (M^e)^d = M^{ed} = M^{k\phi(n)+1} = M\{M^{\phi(n)}\}^k = M$

RSA as a Trapdoor One-way Function

Public key

$$c = f(m) = m^{e} \bmod n$$

Message
m

Ciphertext
c

$$m = f^{-1}(c) = c^{d} \bmod n$$

Private key
(trapdoor information)

n = pq (p & q: primes)

ed = 1 mod (p-1)(q-1)

RSA Public Key Systems

❖ **Example:**

Key Generation

- $p=3$, $q=11$
- $n = pq = 33$, $\phi(n) =(p-1)(q-1) = 2 \times 10 = 20$
- $e = 3$ s.t. $gcd(e, \phi(n))=(3,20)=1$
- Choose d s.t. $ed =1 \bmod \phi(n)$, $3d = 1 \bmod 20$, $d=7$
- Public key $=\{e,n\}=\{3,33\}$, private key $=\{d\}=\{7\}$

Encryption

- $M =5$
- $C = M^e \bmod n = 5^3 \bmod 33 =26$

Decryption

- $M =C^d \bmod n = 26^7 \bmod 33= 5$

RSA Public Key Systems

➢ **Exercise 3: Provide an example of RSA key generation, encryption, and decryption for**

1) p=17, q=23 (by hand calculation)
2) p=2357, q=2551 (using big number calculator)
3) p=885320963, q=238855417 (using big number calculator)

1. Key generation

2. Encryption

3. Decryption

Selecting Primes p and q for RSA

❖ **How to select primes p and q ?**

1. **$|p| \approx |q|$ to avoid ECM (Elliptic Curve Method for factoring)**

2. **p-q must be large to avoid trial division**

3. **p and q are strong prime**
   - **p-1 has large prime factor r (pollard's p-1)**
   - **p+1 has large prime factor (William's p+1)**
   - **r-1 has large prime factor (cyclic attack)**

Security of RSA

❖ **Common Modulus attack:**

❖ **If multiple entities share the same modulus n=pq with different pairs of (e_i, d_i), it is not secure. Do not share the same modulus!**

❖ **Cryptanalysis: If the same message M was encrypted to different users**

User u_1 : $C_1 = M^{e_1} \bmod n$

User u_2 : $C_2 = M^{e_2} \bmod n$

If $\gcd(e_1, e_2)=1$, there are a and b s.t. $ae_1 + be_2 = 1 \bmod n$

Then,

$$(C_1)^a (C_2)^b \bmod n = (M^{e_1})^a (M^{e_2})^b \bmod n = M^{ae_1+be_2} \bmod n = M \bmod n$$

Security of RSA

- **Cycling attack**
  If f(f(…f(M)))=f(M) **where** ***f(M) = M^e mod n ?***
  If a given ciphertext appears after some iterations, we can recover the plaintext at collusion point.
  Let $C=M^e$ mod n
  If $(((C^e)^e)\ldots)^e$ mod n = C^{e^k} mod n = C, then $C^{e^{(k-1)}}$ mod n = M

- **Multiplicative attack (homomorphic property of RSA)**
  $(M_1^e)(M_2^e)$ mod n = $(M_1 \times M_2)^e$ mod n

Attack on RSA Implementations

- **Timing attack: (Kocher 97)**
  The time it takes to compute C^d (mod N) can expose d.

- **Power attack: (Kocher 99)**
  The power consumption of a smartcard while it is computing C^d (mod N) can expose d.

- **Faults attack: (BDL 97)**
  A computer error during C^d (mod N) can expose d.

Security of Public Key Encryption Schemes

- **Security goals**
  - **One-wayness (OW):** the adversary who sees a ciphertext is not able to compute the corresponding message
  - **Indistinguishability (IND):** observing a ciphertext, the adversary learns nothing about the plaintext. Also known as semantic security.
  - **Non-malleability (NM):** observing a ciphertext for a message m, the adversary cannot derive another ciphertext for a meaningful plaintext m' related to m

- **Original RSA encryption is not secure**
  - In IND: deterministic encryption
  - In NM: for example, from $c=m^e$, $c' = 2^e c = (2m)^e$ is easily obtained. It cannot be used in bidding scenario.

Security of Public Key Encryption Schemes

- **Indistinguishability**

m_0, m_1

$b \in_R \{0,1\}$

Challenge: C=E(m_b)

PKE(pk, sk)

Guess b?

The adversary win if he guess *b* correctly with a probability significantly greater than 1/2

Security of Public Key Encryption Schemes

- **Assume the existence of Decryption Oracle**
  - **Mimics an attacker's access to the decryption device**

- **Attack models**
  - **Chosen Plaintext Attack (CPA): the adversary can encrypt any plaintext of his choice. In public key encryption this is always possible.**
  - **Non-adaptive Chosen Ciphertext Attack (CCA1): the attacker has access to the decryption oracle before he sees a ciphertext that he wishes to manipulate**
  - **Adaptive Chosen Ciphertext Attack (CCA2): the attacker has access to the decryption oracle before and after he sees a ciphertext *c* that he wishes to manipulate (but, he is not allowed to query the oracle about the target ciphertext *c*.)**

RSA Padding

- **RSA encryption without padding**
  - Deterministic encryption (same plaintext → same ciphertext)
  - Multiplicative property: $m_1^e \cdot m_2^e = (m_1 m_2)^e \bmod n$
  - Lots of attacks possible
  - Redundancy checking is essential for security

- **RSA encryption with OAEP**
  - RSA encryption after OAEP (Optimal Asymmetric Encryption Padding)
  - Proposed by Bellare and Rogaway
  - Probabilistic encoding of message before encryption
  - RSA becomes a probabilistic encryption
  - Secure against IND-CCA2

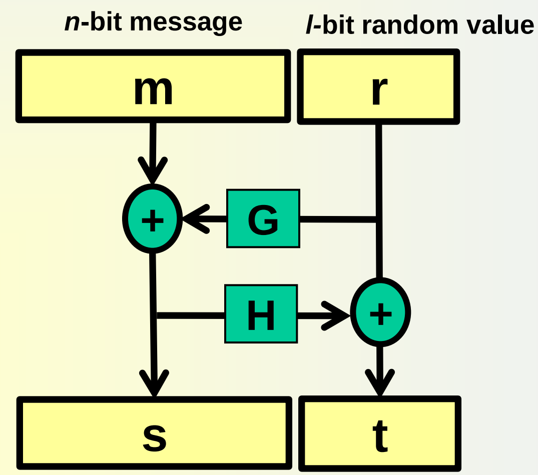
RSA with OAEP

❖ OAEP → RSA encryption

$s = m \oplus G(r)$ $t = r \oplus H(s)$	Encryption padding
$c = E(s,t)$	RSA encryption

❖ RSA decryption → OAEP

$(s,t) = D(c)$	RSA decryption
$r = t \oplus H(s)$ $m = s \oplus G(r)$	Decryption padding



G H Hash function (Random oracle)

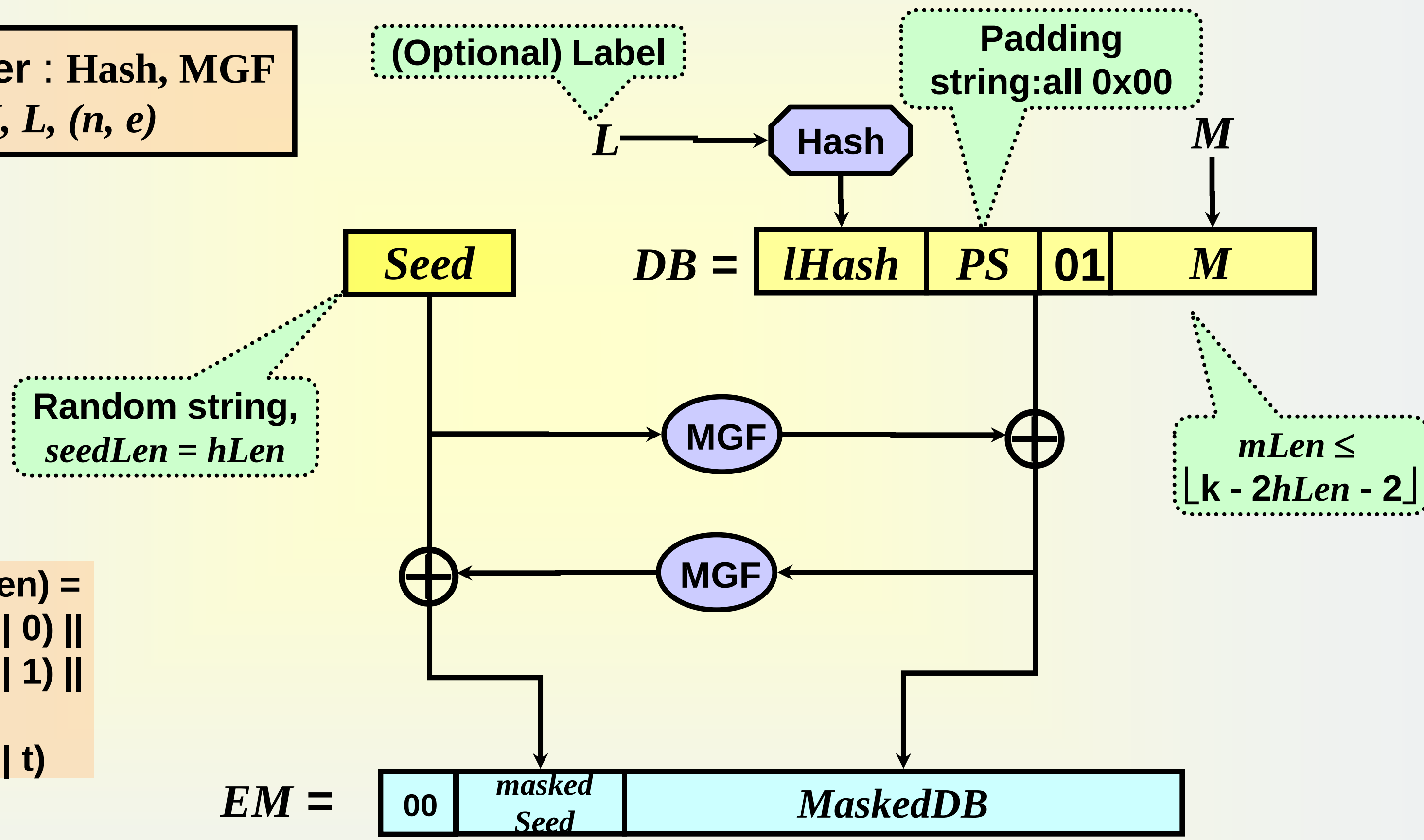
r : *l*-bit random value

OAEP looks like a kind of Feistel network.

RSA Encryption with RSA-OAEP Padding

In PKCS #1 v2.0, v2.1

- **Parameter** : Hash, MGF
- **Input** : M, L, (n, e)



MGF(Seed, Len) =
Hash(Seed || 0) ||
Hash(Seed || 1) ||
. . .
Hash(Seed || t)

$$C = (EM)^e \bmod n$$

Diffie-Hellman / ElGamal-type Systems

- **Domain parameter generation**
  - Based on the hardness of DLP
  - Generate a large (1024 bits or more) prime p
  - Find a generator g that generates the cyclic group Z_p^*
  - Domain parameter = {p, g}

- **Key generation**
  - Pick a random integer $x \in [1, p-1]$
  - Compute $y = g^x \bmod p$
  - Public key (p, g, y) : publish
  - Private key x : keep secret

- **Applications**
  - Public key encryption
  - Digital signatures
  - Key agreement

ElGamal Encryption Scheme

- **Keys & parameters**
  - **Domain parameter = {p, g}**
  - **Choose $x \in [1, p-1]$ and compute $y = g^x \bmod p$**
  - **Public key (p, g, y)**
  - **Private key x**

- **Encryption: $m \rightarrow (C_1, C_2)$**
  - **Pick a random integer $k \in [1, p-1]$**
  - **Compute $C_1 = g^k \bmod p$**
  - **Compute $C_2 = m \times y^k \bmod p$**

- **Decryption**
  - **$m = C_2 \times C_1^{-x} \bmod p$**
  - **$C_2 \times C_1^{-x} = (m \times y^k) \times (g^k)^{-x} = m \times (g^x)^k \times (g^k)^{-x} = m \bmod p$**

ElGamal Encryption Scheme -- Example

- **Key Generation**
  - **Let p=23, g=7**
  - **Private key x=9**
  - **Public key $y = g^x \bmod p = 7^9 \bmod 23 = 15$**

- **Encryption: m → (C_1, C_2)**
  - **Let m=20**
  - **Pick a random number k=3**
  - **Compute $C_1 = g^k \bmod p = 7^3 \bmod 23 = 21$**
  - **Compute $C_2 = m \times y^k \bmod p = 20 \times 15^3 \bmod 23 = 20 \times 17 \bmod 23 = 18$**
  - **Send $(C_1, C_2) = (21,18)$ as a ciphertext**

- **Decryption**
  - **$m = C_2 / C_1^x \bmod p = 18 / 21^9 \bmod 23 = 18 / 17 \bmod 23 = 20$**

4. Digital Signatures

RSA, ElGamal, DSA, KCDSA, Schnorr

Digital Signature

- Digital Signature
  - Electronic version of handwritten signature on electronic document
  - Signing using private key (only by the signer)
  - Verification using public key (by everyone)

- Hash then sign: sig(h(m))
  - Efficiency in computation and communication

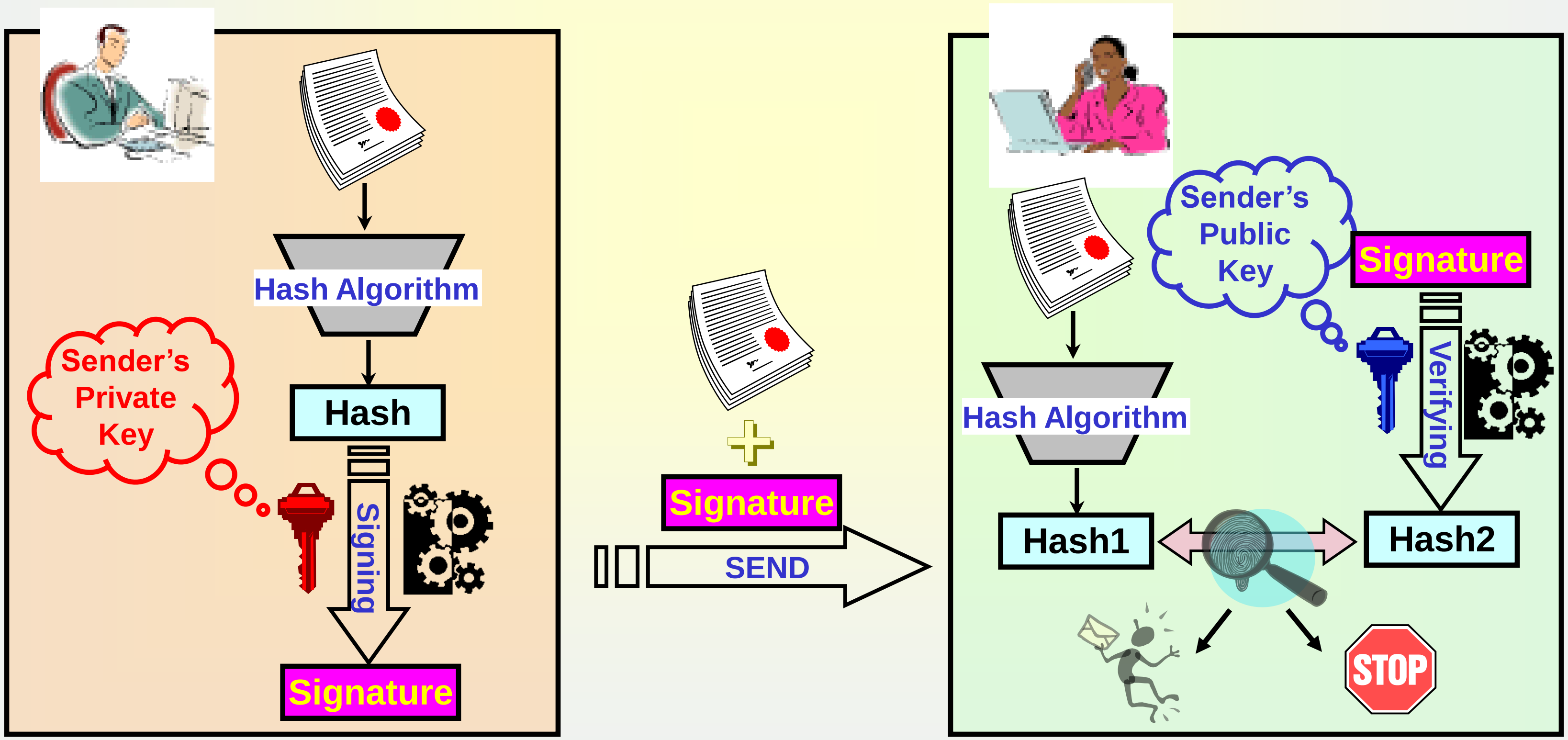
Digital Signature

- ❖ **Security requirements for digital signature**
  - ➢ **Unforgeability (위조 방지)**
  - ➢ **User authentication (사용자 인증)**
  - ➢ **Non-repudiation (부인 방지)**
  - ➢ **Unalterability (변조 방지)**
  - ➢ **Non-reusability (재사용 방지)**

- ❖ **Services provided by digital signature**
  - ❖ **Authentication**
  - ❖ **Data integrity**
  - ❖ **Non-Repudiation**

Digital Signature

- **Digital Signature**
  - ✓ Combine Hash with Digital Signature and use PKC
  - ✓ Provide Authentication and Non-Repudiation
  - ✓ RSA; DSA, KCDSA, ECDSA, EC-KCDSA



RSA Signature

- **Key generation**
  - Choose two large (512 bits or more) primes p & q
  - Compute modulus $n = pq$, and $\phi(n) = (p-1)(q-1)$
  - Pick an integer e relatively prime to $\phi(n)$, $\gcd(e, \phi(n))=1$
  - Compute d such that $ed = 1 \bmod \phi(n)$
  - **Public key (n, e)** : publish
  - **Private key d** : keep secret (may discard p & q)

- **Signing / Verifying**
  - S: $s = m^d \bmod n$ for $0 < m < n$
  - V: $m =? s^e \bmod n$
  - S: $s = h(m)^d \bmod n$ --- hashed version
  - V: $h(m) =? s^e \bmod n$

- **RSA signature without padding**
  - Deterministic signature, no randomness introduced

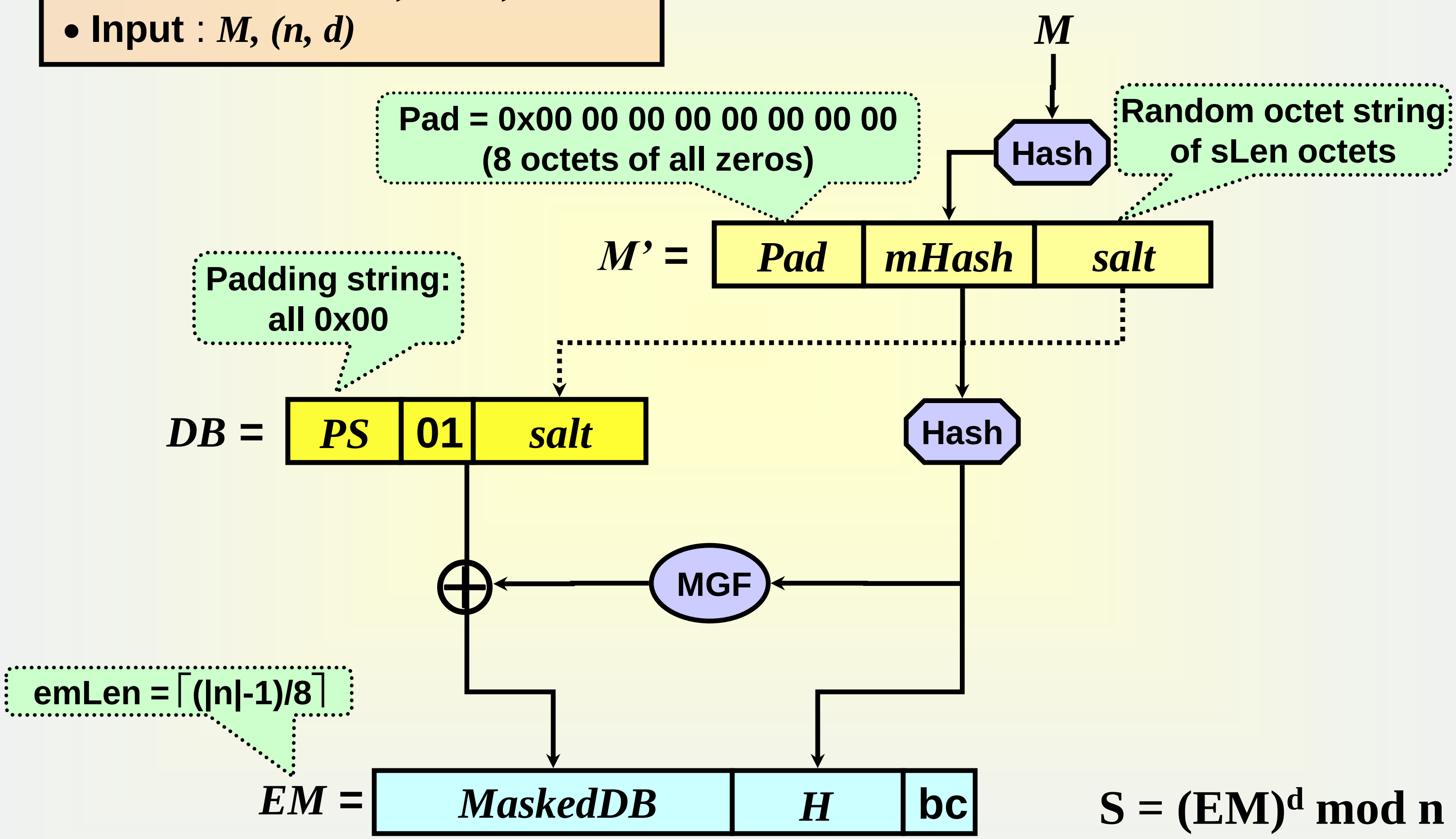
RSA Signature

- **RSA signature forgery: Attack based on the multiplicative property of RSA.**
  $y_1 = (m_1)^d$
  $y_2 = (m_2)^d$, **then**
  $(y_1 y_2)^e = m_1 m_2$
  Thus $y_1 y_2$ **is a valid signature of** $m_1 m_2$

  This is an existential forgery using a known message attack.

RSA Signing with RSA-PSS Padding

- **Parameter** : Hash, MGF, *sLen*
- **Input** : *M, (n, d)*



$S = (EM)^d \bmod n$

ElGamal Signature Scheme

- **Keys & parameters**
  - Domain parameter = {p, g}
  - Choose $x \in [1, p-1]$ and compute $y = g^x \bmod p$
  - Public key (p, g, y)
  - Private key x

- **Signature generation: (r, s)**
  - Pick a random integer $k \in [1, p-1]$
  - Compute $r = g^k \bmod p$
  - Compute s such that $m = xr + ks \bmod p-1$

- **Signature verification**
  - $y^r r^s \bmod p \overset{?}{=} g^m \bmod p$
    - If equal, accept the signature (valid)
    - If not equal, reject the signature (invalid)

- **No hash function…**

Digital Signature Algorithm (DSA)

Private : x
Public : p, q, g, y

p : 512 ~ 1024-bit prime
q : 160-bit prime, $q \mid p-1$
g : generator of order q
x : $0 < x < q$
$y = g^x \bmod p$

- **Signing**

  Pick a random k s.t. $0 < k < q$

  $$r = (g^k \bmod p) \bmod q$$
  $$s = k^{-1}(\mathrm{SHA1}(m) + xr) \bmod q$$

  $m, (r,s) \Rightarrow$

- **Verifying**

  $m, (r,s) \Rightarrow$

  $$w = s^{-1} \bmod q$$
  $$u1 = \mathrm{SHA1}(m) \times w \bmod q$$
  $$u2 = r \times w \bmod q$$
  $$v = (g^{u1} \times y^{u2} \bmod p) \bmod q$$
  $$v =? \, r$$

Korean Certificate-based Digital Signature Algorithm (KCDSA)

Private : x
Public : p, q, g, y
$z = h(Cert\_Data)$

p : 768+256k (k=0 ~ 5) bit prime
q : 160+32k (k=0~3) bit prime, $q \mid p-1$
g : generator of order q
x : $0 < x < q$
$y = g^{x'} \bmod p,\ x' = x^{-1} \bmod q$

- **Signing**

Pick a random k s.t. $0 < k < q$

$$r = \text{HAS160}(g^k \bmod p)$$
$$e = r \oplus \text{HAS160}(z \,\|\, m)$$
$$s = x(k - e) \bmod q$$

$m, (r,s)$ ⇒

- **Verifying**

$m, (r,s)$ ⇒

$$e = r \oplus \text{HAS160}(z \,\|\, m)$$
$$v = y^s \times g^e \bmod p$$
$$\text{HAS160}(v) =?\ r$$

Schnorr Signature Scheme

- **Domain parameters**
  - p = a large prime (~ size 1024 bit), q = a prime (~size 160 bit)
  - q = a large prime divisor of $p-1$ ($q \mid p-1$)
  - g = an element of Z_p of order q, i.e., $g \neq 1$ & $g^q = 1 \bmod p$
  - Considered in a subgroup of order q in modulo p
- **Keys**
  - Private key $x \in_R [1, q-1]$: a random integer
  - Public key $y = g^x \bmod p$
- **Signature generation**: (r, s)
  - Pick a random integer $k \in_R [1, q-1]$
  - Compute $r = h(g^k \bmod p, m)$
  - Compute $s = k - xr \bmod q$
- **Signature verification**
  - $r =? h(y^r g^s \bmod p, m)$

Security of Digital Signature Schemes

- Security goals
  - **Total break**: adversary is able to find the secret for signing, so he can forge then any signature on any message.
  - **Selective forgery**: adversary is able to create valid signatures on a message chosen by someone else, with a significant probability.
  - **Existential forgery**: adversary can create a pair (message, signature), s.t. the signature of the message is valid.

Security of Digital Signature Schemes

- **Attack models**
  - **Key-only attack: Adversary knows only the verification function (which is supposed to be public).**
  - **Known message attack: Adversary knows a list of messages previously signed by Alice.**
  - **Chosen message attack: Adversary can choose what messages wants Alice to sign, and he knows both the messages and the corresponding signatures.**

5. Signcryption

Signature + Encryption

What is Signcryption?

- Provides the functions of
  - digital signature : unforgeability & non-repudiation
  - public key encryption : confidentiality

- Two birds in one stone

- Has a significantly smaller computation & communication cost compared with traditional digital envelop (signature-then-encryption)

Cost (signcryption) << Cost (signature) + Cost (encryption)

Signcryption – system setup

- Public to all
  - *p* : a large prime
  - *q* : a large prime factor of *p-1*
  - *g* : *0<g<p* & with order *q* mod *p*
  - *G*: 1-way hash
  - *H*: 1-way hash
  - (*E*,*D*) : symmetric key encryption & decryption algorithms

Alice's keys:

x_a : secret key

y_a : public key

$(y_a = g^{x_a} \bmod p)$

Bob's keys:

x_b : secret key

y_b : public key

$(y_b = g^{x_b} \bmod p)$

Signcryption – 1st Example

$$m \longrightarrow (c,r,s) \qquad\qquad (c,r,s) \longrightarrow m$$

Signcryption by Alice :

1. Pick at random $x \in_R \{1,\ldots,q-1\}$
2. $w = y_b^x \bmod p$
3. $k = G(w)$
4. $r = H(m, bind\_info, w)$
5. $s = x/(r + x_a) \bmod q$
6. $c = E_k(m)$
7. return (c, r, s)

Unsigncryption by Bob :

1. $w = (y_a \cdot g^r)^{s \cdot x_b} \bmod p$
2. $k = G(w)$
3. $m = D_k(c)$
4. Return m if

   $r = H(m, bind\_info, w)$
5. Return "invalid" otherwise

Signcryption – 2nd Example

$$\boldsymbol{m} \longrightarrow \boldsymbol{(c,r,s)}$$

Signcryption by Alice :

1. Pick at random $x \in_R \{1, \ldots, q-1\}$
2. $w = y_b^x \bmod p$
3. $k = G(w)$
4. $r = H(m, bind\_info, w)$
5. $\boxed{s = x/(1 + x_a \cdot r) \bmod q}$
6. $c = E_k(m)$
7. return (c, r, s)

$$\boldsymbol{(c,r,s)} \longrightarrow \boldsymbol{m}$$

Unsigncryption by Bob :

1. $\boxed{w = (g \cdot y_a{}^r)^{s \cdot x_b} \bmod p}$
2. $k = G(w)$
3. $m = D_k(c)$
4. Return m if

   $r = H(m, bind\_info, w)$
5. Return "invalid" otherwise

Signcryption – 3rd Example

$\boldsymbol{m} \longrightarrow \boldsymbol{(c,r,s)}$

Signcryption by Alice :

1. Pick at random $x \in_R \{1,\dots,q-1\}$
2. $w = y_b^x \bmod p$
3. $k = G(w)$
4. $r = H(m, bind\_info, w)$
5. $\boxed{s = (x - x_a \cdot r) \bmod q}$
6. $c = E_k(m)$
7. return (c, r, s)

$\boldsymbol{(c,r,s)} \longrightarrow \boldsymbol{m}$

Unsigncryption by Bob :

1. $\boxed{w = (g^s \cdot y_a^r)^{x_b} \bmod p}$
2. $k = G(w)$
3. $m = D_k(c)$
4. Return m if

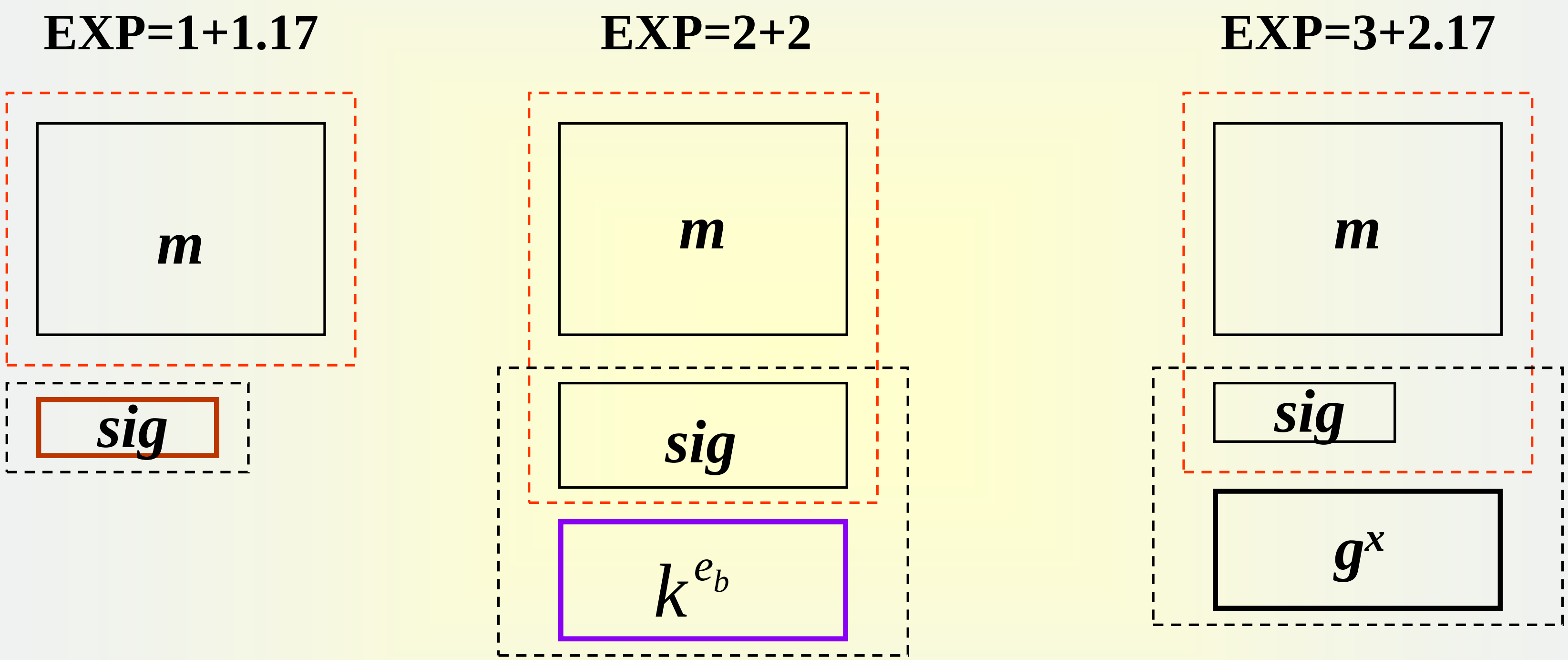
   $r = H(m, bind\_info, w)$
5. Return "invalid" otherwise

Major Instantiations of Signcryption

- based on DL on an Elliptic Curve
  - Zheng, CRYPTO'97
  - Zheng & Imai IPL 1998
- based on other sub-groups (e.g. XTR)
  - Lenstra & Verheul, CRYPTO2000
  - Gong & Harn, IEEE-IT 2000
  - Zheng, CRYPTO'97
- based on DL on finite field
  - Zheng, CRYPTO'97
- based on factoring / residuosity
  - Steinfeld & Zheng, ISW2000
  - Zheng, PKC2001

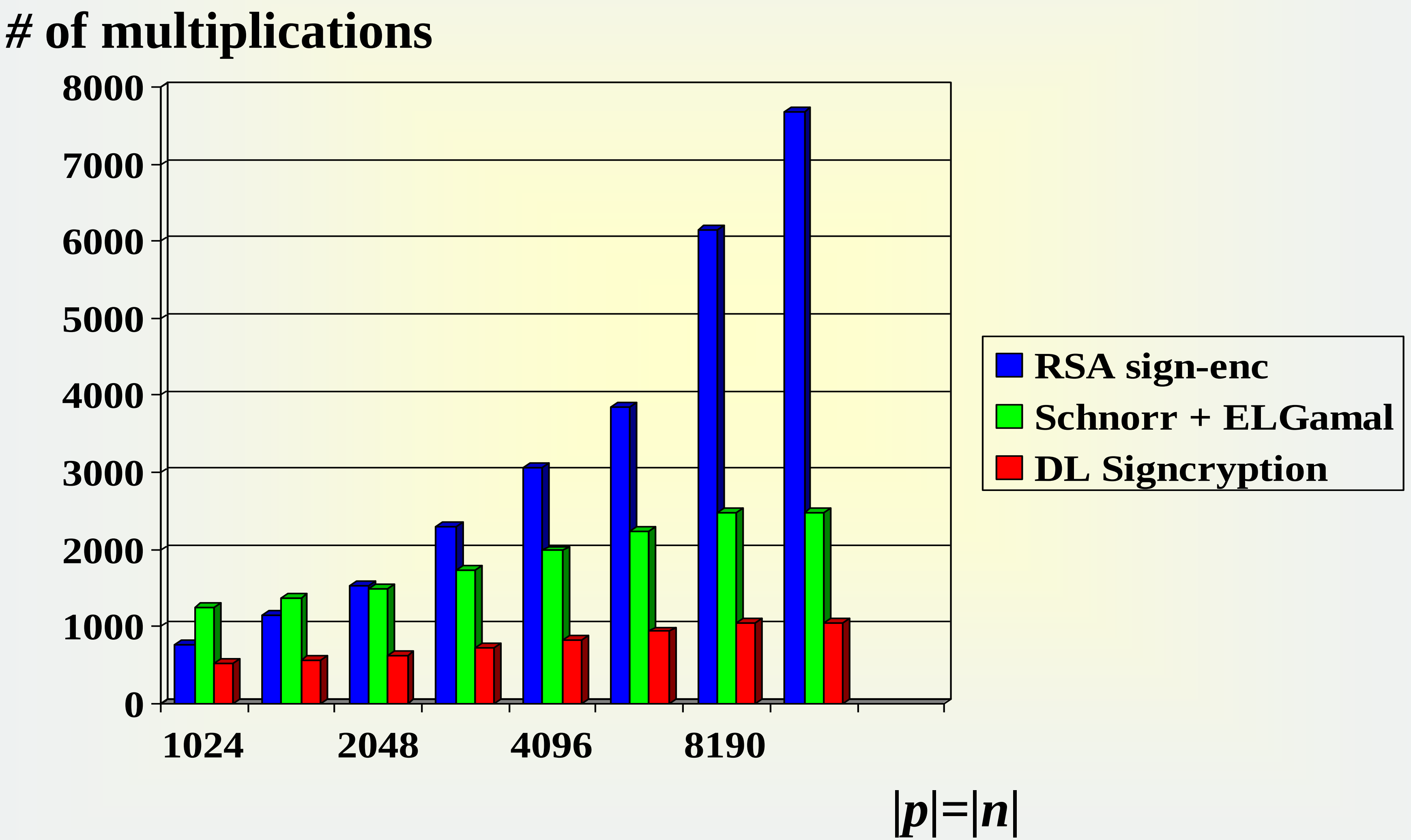
More efficient

Signcryption vs. Signature-then-Encryption

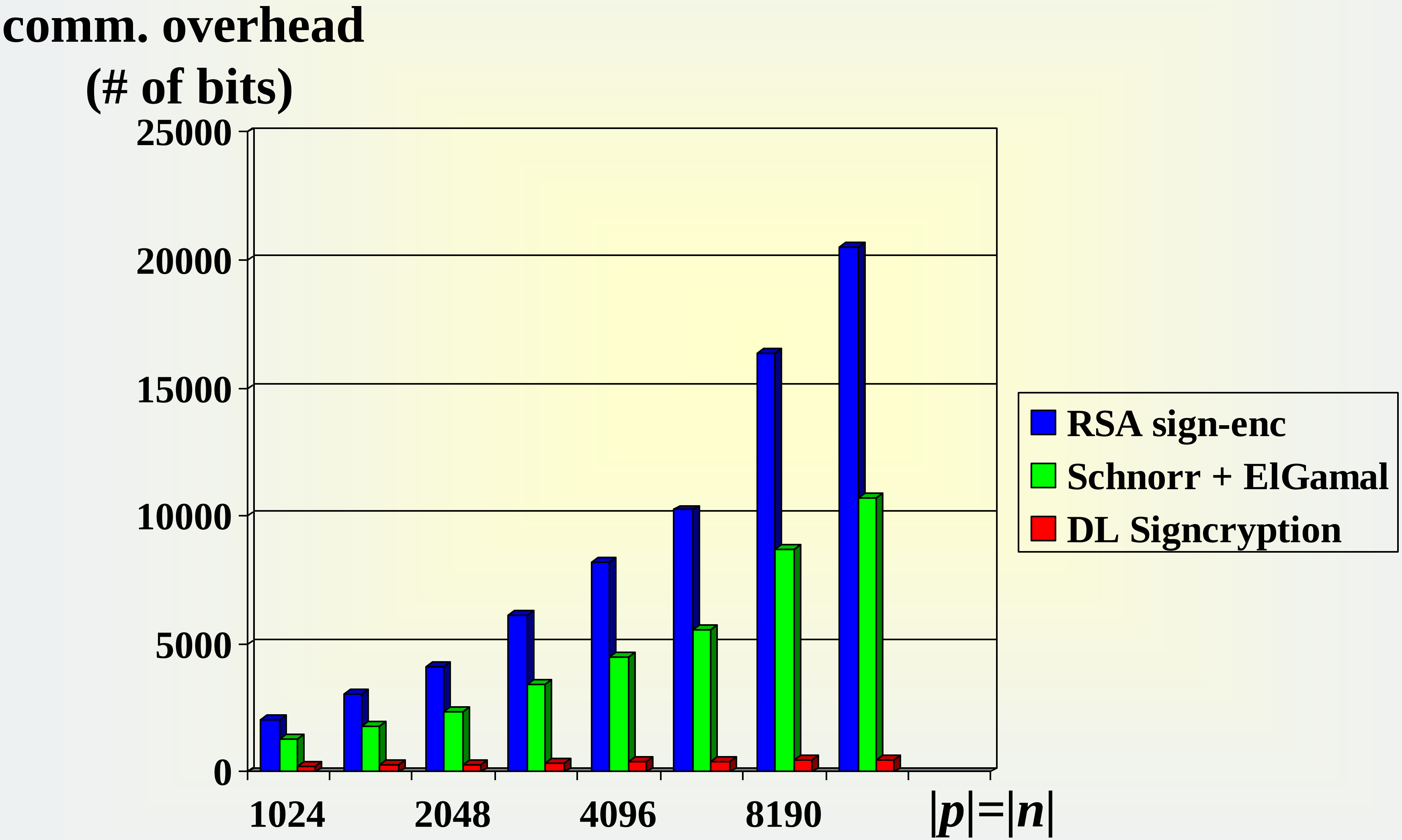


(a) Signcryption based on DL **(b) Signature-then-Encryption based on RSA** **(c) Signature-then-Encryption based on DL**

DL-based Signcryption vs. Signature-then-Encryption



DL-based Signcryption vs. Signature-then-Encryption



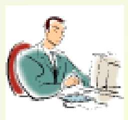
Security Proofs

- Proofs for the confidentiality and unforgeability of signcryption
  - Confidentiality --- Providing a reduction
    - from breaking the security of signcryption with respect to adaptive chosen ciphertext attacks in the flexible public key model
    - to breaking the GAP Diffie-Hellman assumption, in the random oracle model
  - Unforgeability --- Providing a reduction
    - from breaking the unforgeability of signcryption against adaptive chosen message attacks
    - to the Discrete Logarithm problem, in the random oracle model

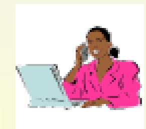
6. Key Exchange

Diffie-Hellman

Diffie-Hellman Key Agreement Scheme



Domain Parameters
p, g



choose $X_a \in [1, p-1]$

$Y_a = g^{X_a} \bmod p$

choose $X_b \in [1, p-1]$

$Y_b = g^{X_b} \bmod p$

$$Y_a \longrightarrow$$

$$\longleftarrow Y_b$$

compute the shared key

$K_a = Y_b^{X_a} = g^{X_b X_a} \bmod p$

compute the shared key

$K_b = Y_a^{X_b} = g^{X_a X_b} \bmod p$

Diffie-Hellman Problem

- **Computational Diffie-Hellman (CDH) Problem**

  Given $Y_a = g^{X_a} \bmod p$ and $Y_b = g^{X_b} \bmod p$,

  compute $K_{ab} = g^{X_a X_b} \bmod p$

- **Decision Diffie-Hellman (DDH) Problem**

  Given $Y_a = g^{X_a} \bmod p$ and $Y_b = g^{X_b} \bmod p$,

  distinguish between $K_{ab} = g^{X_a X_b} \bmod p$ and a random string

- **Discrete Logarithm Problem (DLP)**

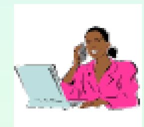
  Given $Y = g^X \bmod p$, compute $X = \log_b Y$.

The Security of the Diffie-Hellman key agreement depends on the difficulty of CDH problem.

Man in the Middle Attack in Diffie-Hellman Key Agreement

X_b : private

$Y_b = g^{X_b}$: public



X_a : private

$Y_a = g^{X_a}$: public

Y_b →

← Y_c

$Y_c = g^{X_c}$ for some X_c

Y_c →

← Y_a

Bob computes the session key

$K_b = Y_c^{X_b} = g^{X_c X_b}$

Alice computes the session key

$K_a = Y_c^{X_a} = g^{X_c X_a}$

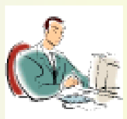
Adversary computes the both session keys

$K_b = Y_b^{X_c} = g^{X_c X_b}$

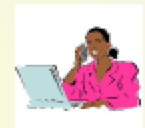
$K_a = Y_a^{X_c} = g^{X_c X_a}$

Problem comes from no authentication

Diffie-Hellman Key Agreement using Certified Key



Domain Parameters
p, g



choose $X_a \in [1, p-1]$

$Y_a = g^{X_a} \bmod p$

Certified key
Y_a and Y_b

choose $X_b \in [1, p-1]$

$Y_b = g^{X_b} \bmod p$

compute the shared key

$K_a = Y_b^{X_a} = g^{X_b X_a} \bmod p$

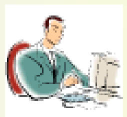
compute the shared key

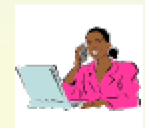
$K_b = Y_a^{X_b} = g^{X_a X_b} \bmod p$

- Interaction is not required
- Agreed key is fixed, long-term use

MTI Protocols -- by Matsumoto, Takashima, Imai

Domain Parameters
p, g





choose $X_a \in [1, p-1]$
$Y_a = g^{X_a} \mod p$

Certified key
Y_a and Y_b

choose $X_b \in [1, p-1]$
$Y_b = g^{X_b} \mod p$

Choose $k_a \in [1, p-1]$
$T_a = g^{k_a} \mod p$

$\xrightarrow{T_a}$

$\xleftarrow[T_b]{}$

choose $k_b \in [1, p-1]$
$T_b = g^{k_b} \mod p$

compute the shared key
$K_a = Y_b^{k_a} T_b^{X_a} = g^{X_b k_a} g^{k_b X_a}$

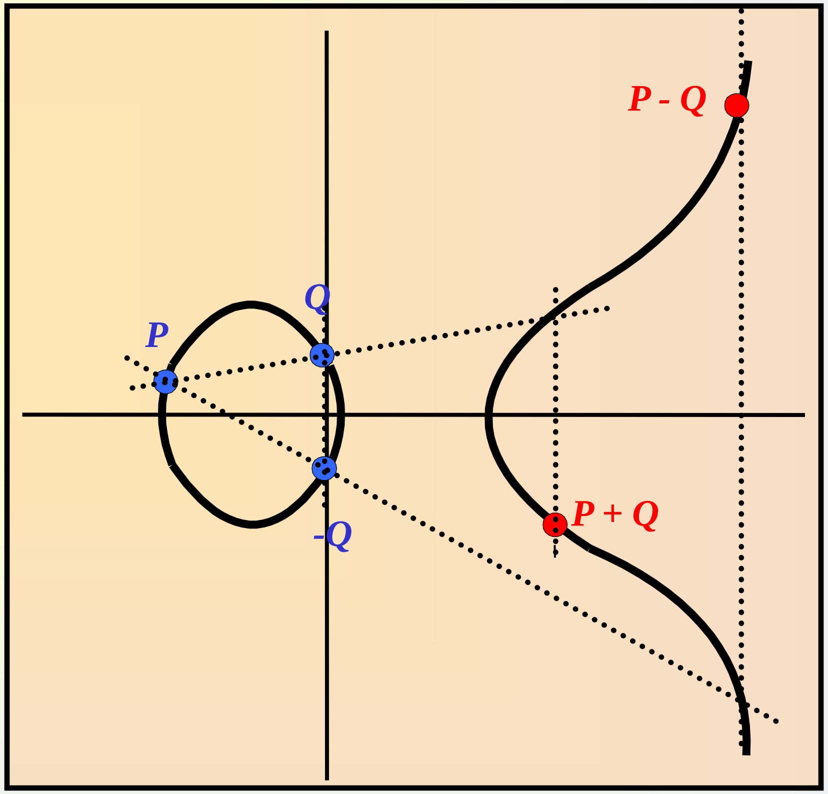
compute the shared key
$K_b = Y_a^{k_b} T_a^{X_b} = g^{X_a k_b} g^{k_a X_b}$

7. Elliptic Curve Cryptosystem

Elliptic Curve (1)

- Weierstrass form of Elliptic Curve
  - ✓ $y^2 + a_1 xy + a_3 = x^3 + a_2 x^2 + a_4 x + a_6$
- Example (over rational field)
  - ✓ $y^2 = x^3 – 4x + 1$
  - ✓ E(Q)

    $= \{(x,y) \in Q^2 \mid y^2 = x^3 – 2x + 2\} \cup O_E$
  - ✓ P = (2, 1), –P = (2, –1)
  - ✓ [2]P = (12 , -41)
  - ✓ [3]P = (91/25, 736/125)
  - ✓ [4]P = (5452/1681, -324319/68921)



Elliptic Curve (2)

- Example (over finite field GF(p) : p = 13)
  - ✓ P = (2,1), –P = (2, 12), [2]P = (12, 11)
  - ✓ [3]P = (0, 1), [4]P = (11, 12), …… , [18]P = O_E
  - ✓ Hasse's Theorem : $p - 2\sqrt{p} \leq$ # of E(p) $\leq p + 2\sqrt{p}$
  - ✓ Scalar multiplication: [d]P
- Elliptic Curve Discrete Logarithm
  - ✓ Base of Elliptic Curve Cryptosystem (ECC)

$$\boxed{y = g^x \bmod p} \Longleftrightarrow \boxed{Q = [d]P}$$

Find x for given Y　　Find d for given Q

Elliptic Curve Cryptosystems

- **Advantages**
  - Breaking PKC over Elliptic Curve is much harder
  - We can use much shorter key
  - Encryption/Decryption is much faster than that of other PKCs
  - It is suitable for restricted environments like mobile phone, smart card

- **Disadvantages**
  - It's new technique ➔ There may be new attacks
  - Too complex to understand
  - ECC is a minefield of patents

    : e.g. US patents

    4587627/739220 – Normal Basis, 5272755 – Curve over GF(p)

    5463690/5271051/5159632 – p=2^q-c for small c, etc…

Key Sizes and Algorithms

- **System strength, Symmetric Key strength, Public Key strength** must be consistently matched for any network protocol usage.
- **Selection Rules**
  - ✓ Determine symmetric key sizes : n
  - ✓ Symmetric Cipher → Key exchange Algorithm → Authentication Algorithm

Sym.	RSA/DH	ECC
64	512	-
90	1024	160
120	2048	210
128	2304	256

From Peter Gutmann's tutorial

Sym.	RSA/DH	ECC
56	430	112
80	760	160
96	1020	192
128	1620	256

From RSA's Bulletin (2000. 4. No 13)

- **Recommendation for RSA/ECC**
  - ✓ 512/112-bit : only for micropayment/smart card
  - ✓ 1024/160-bit : for short term (1-year) security
  - ✓ 2048/256-bit : for long term security (CA,RA)

Implementation Results

- **RSA Encryption/Decryption**

	Encryption	Decryption
PKCS#1-v1.5	1.49 ms	18.05 ms
PKCS#1-OAEP	1.41 ms	18.09 ms

- **Signature**

	Signing	Verifying
PKCS#1-v1.5	18.07 ms	1.24 ms
PKCS#1-PSS	18.24 ms	1.28 ms
DSA with SHA1	2.75 ms	9.85 ms
KCDSA with HAS160	2.42 ms	9.55 ms

- **Modular Exponentiation vs. Scalar Multiplication of EC**

M.E. (1024-bit)	S.M. ($GF(2^{162})$)	S.M. (GF(p))
52.01 ms	2.24 ms	1.17 ms

Implementation Environments

- **RSA Encryption/Signature**
  - ✓ N : 1024 bits, public exponent : $65537 = 2^{16} + 1$
  - ✓ Decryption/Signing uses Chinese Remainder Theorem (CRT)

    : CRT is roughly 3 times faster

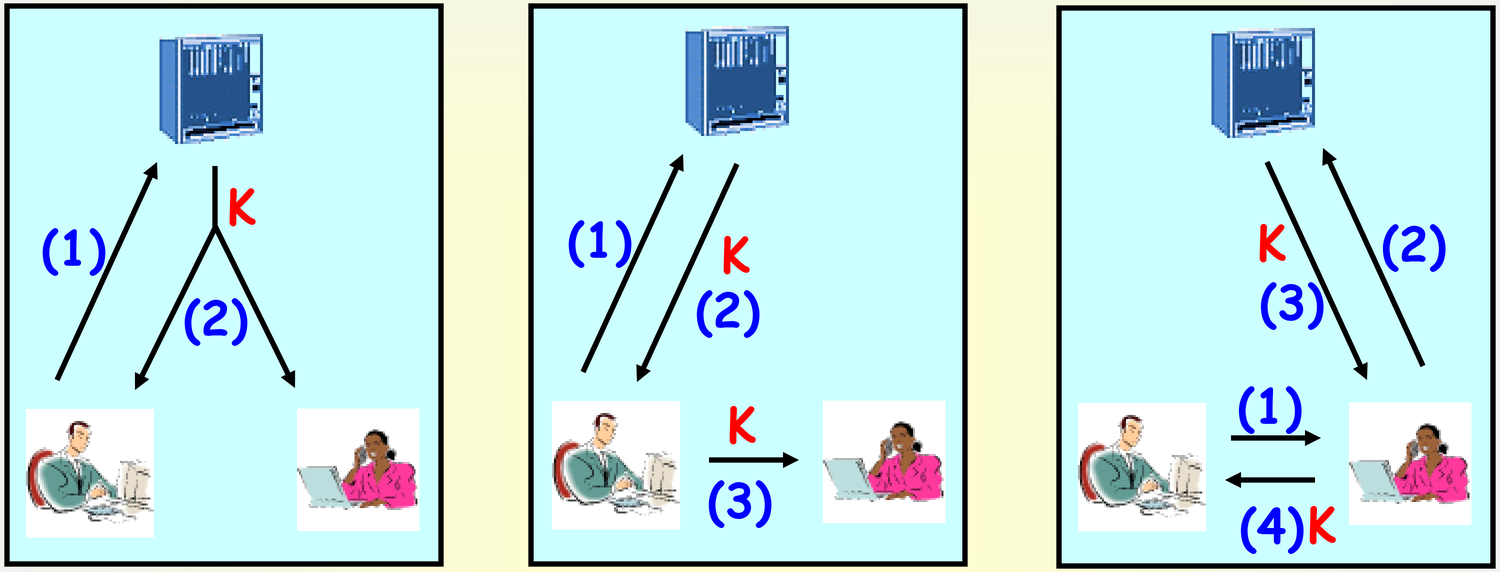
PIII 450MHz
Widows 98
MSVC++ 6.0
with assembly

- **DSA/KCDSA**
  - ✓ p : 1024-bit prime, q : 160-bit subprime
  - ✓ Signing uses LL-method
  - ✓ Verifying uses double-exponentiation

- **Modular Exponentiation vs. Scalar Multiplication of EC**
  - ✓ M.E./S.M. uses Window-method
  - ✓ In the same security level, ECC is much faster that RSA/DSA

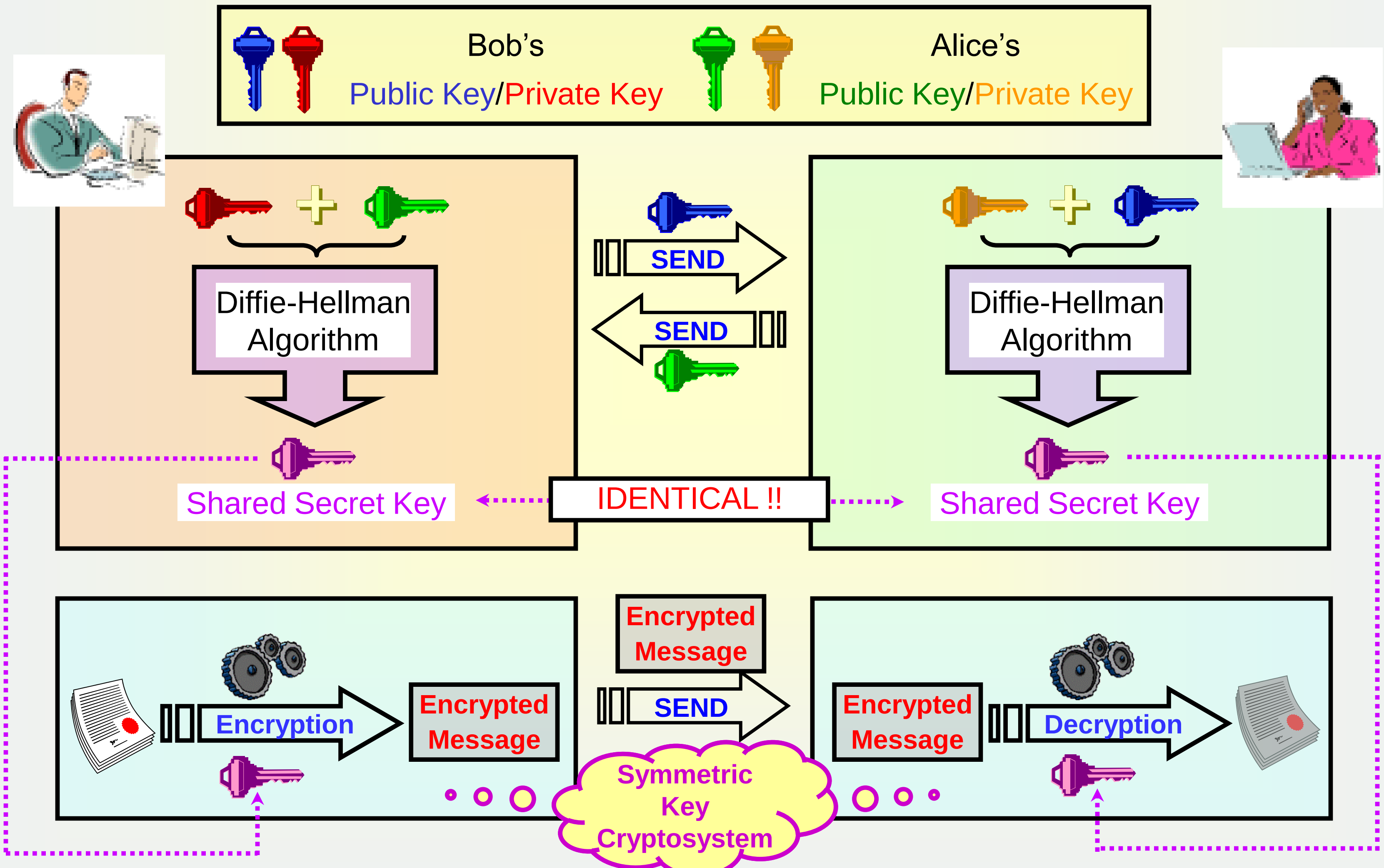
8. Certification and PKI

Key Distribution Center (KDC)

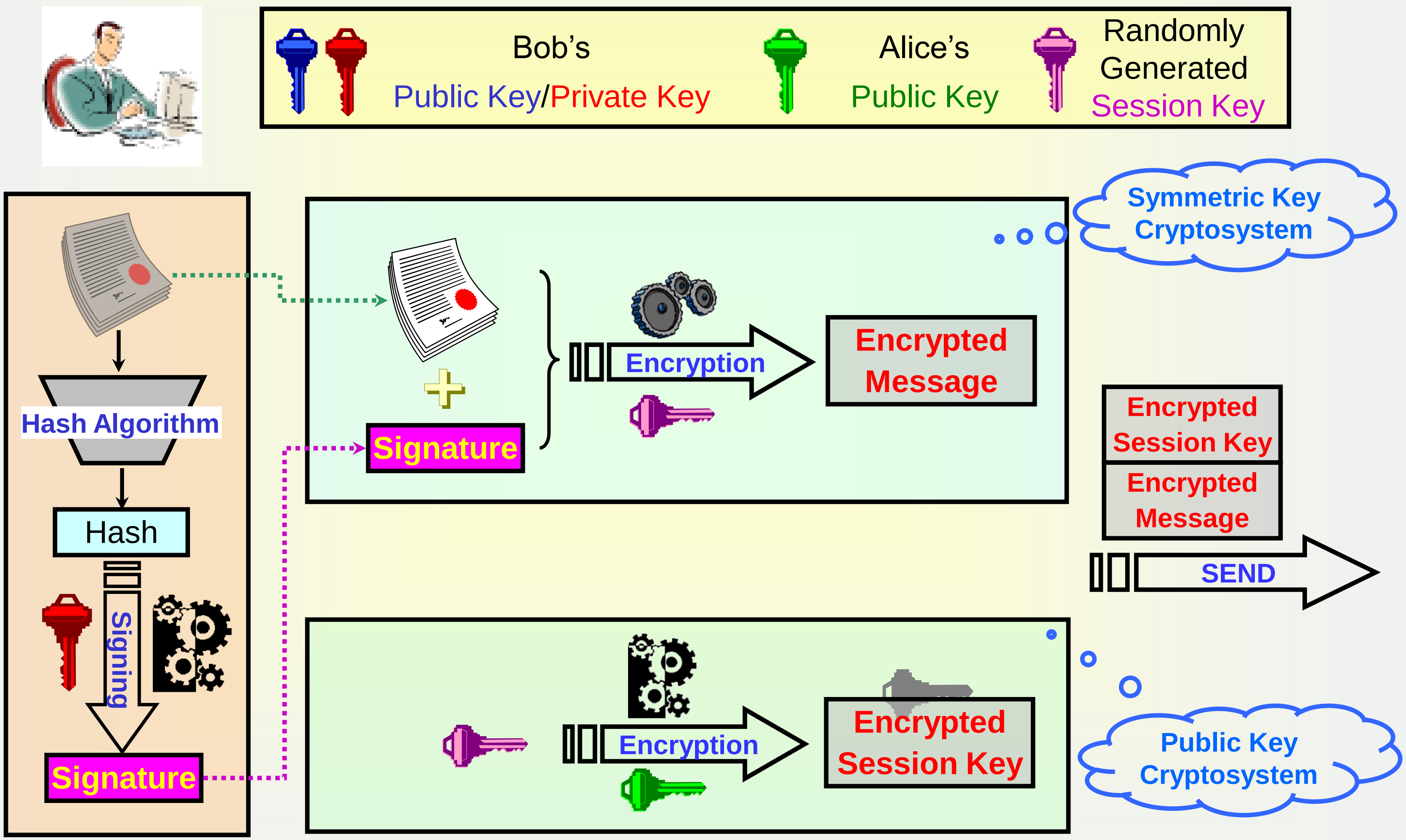


- ✓ **Rely on the absolute security of KDC**
- ✓ **Ease of centralized management**
- ✓ **Suitable for enterprise network security**
- ✓ **But not Scalable; KDC is a potential Bottleneck**

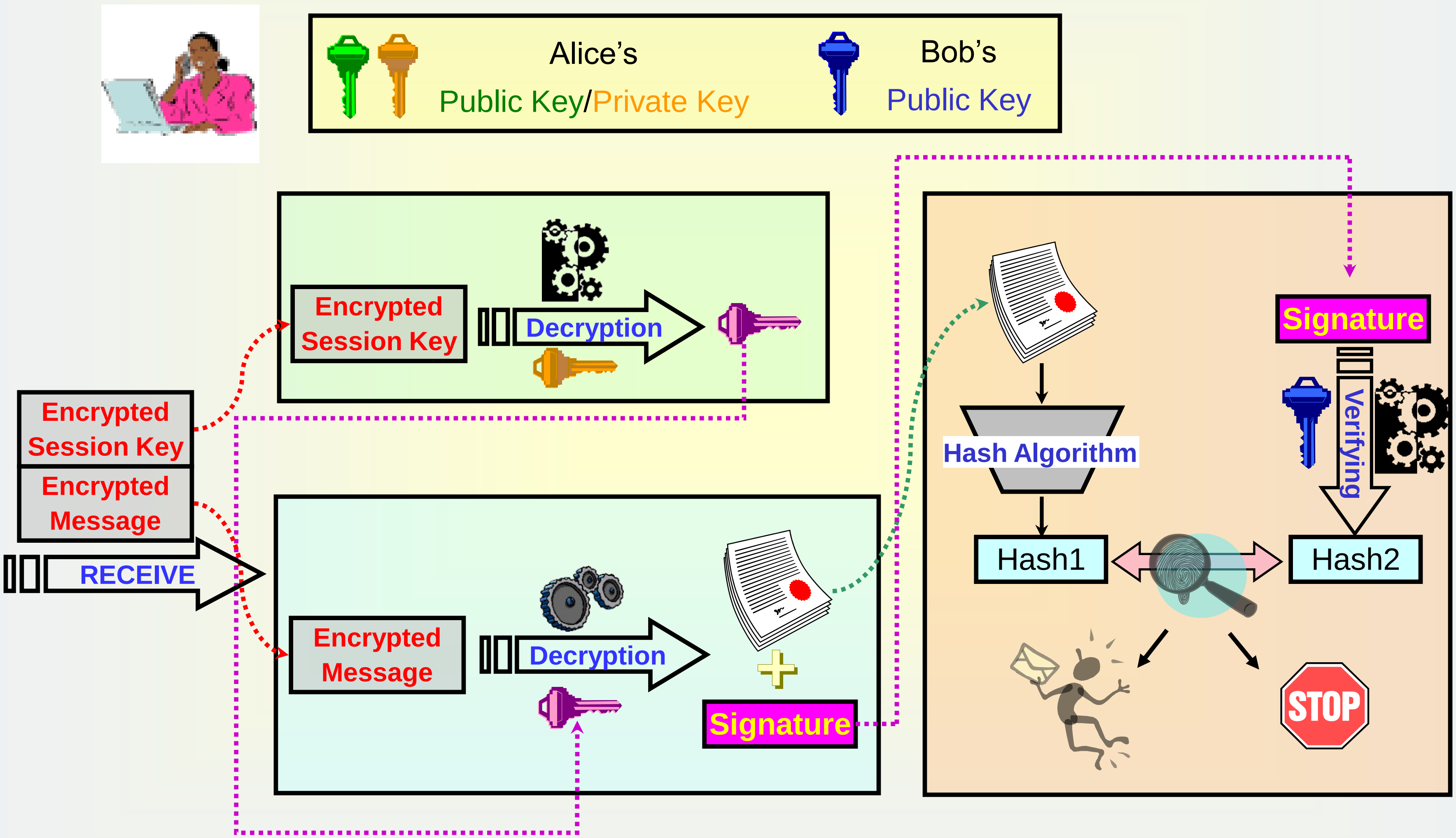
Diffie-Hellman Key Exchange and Message Encryption



Digital Enveloping : Key Transport + Encryption



Digital Enveloping : Key Recovery + Decryption



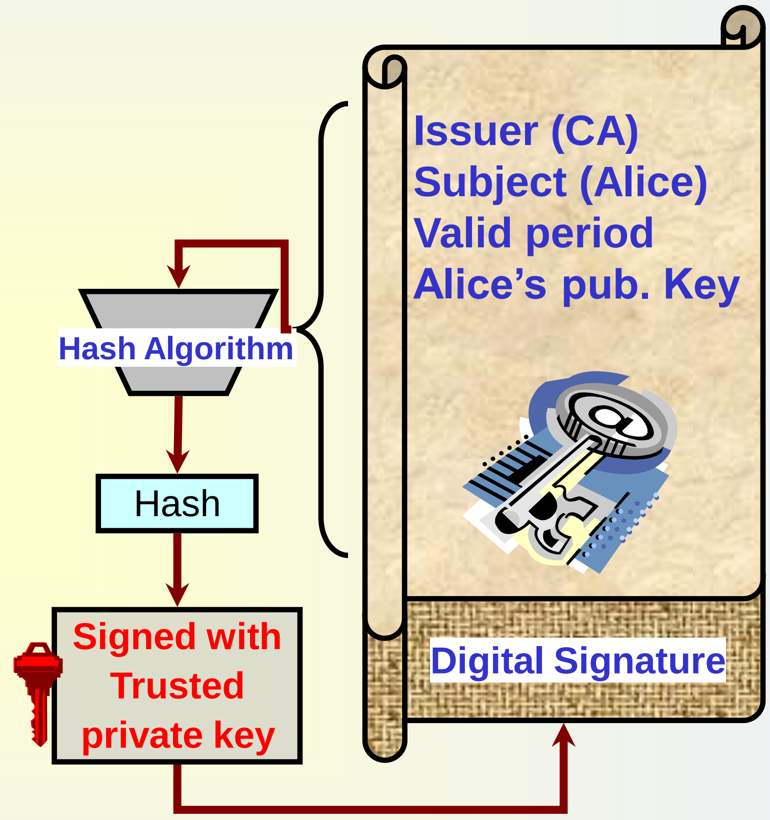
How to Guarantee Authenticity of Peer Public Key?

- For secure use of public key systems,
  - Everyone should be able to obtain the public key of any communication peer that he wants to communicate with, in such a way that he can be sure that the obtained public key is the correct and right public key of the peer
  - How to guarantee that the public key obtained is the right one ?
  - How to guarantee that the public key obtained is authentic ?
- Using Certificate !

What is a Digital Certificate?

❖ **Digital Certificate**

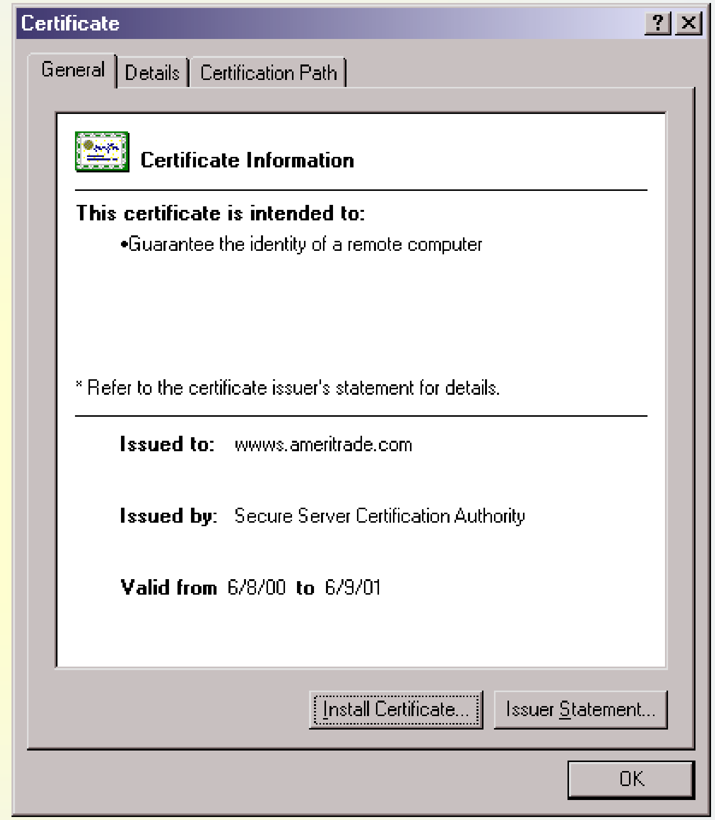
- ✓ A file containing Identification information (CA's name (Issuer), Alice's name (Subject), valid period, Alice's public key, etc) and digital signature signed by trusted third (CA) to guarantee its authenticity & integrity



❖ **Certificate Authority (CA)**

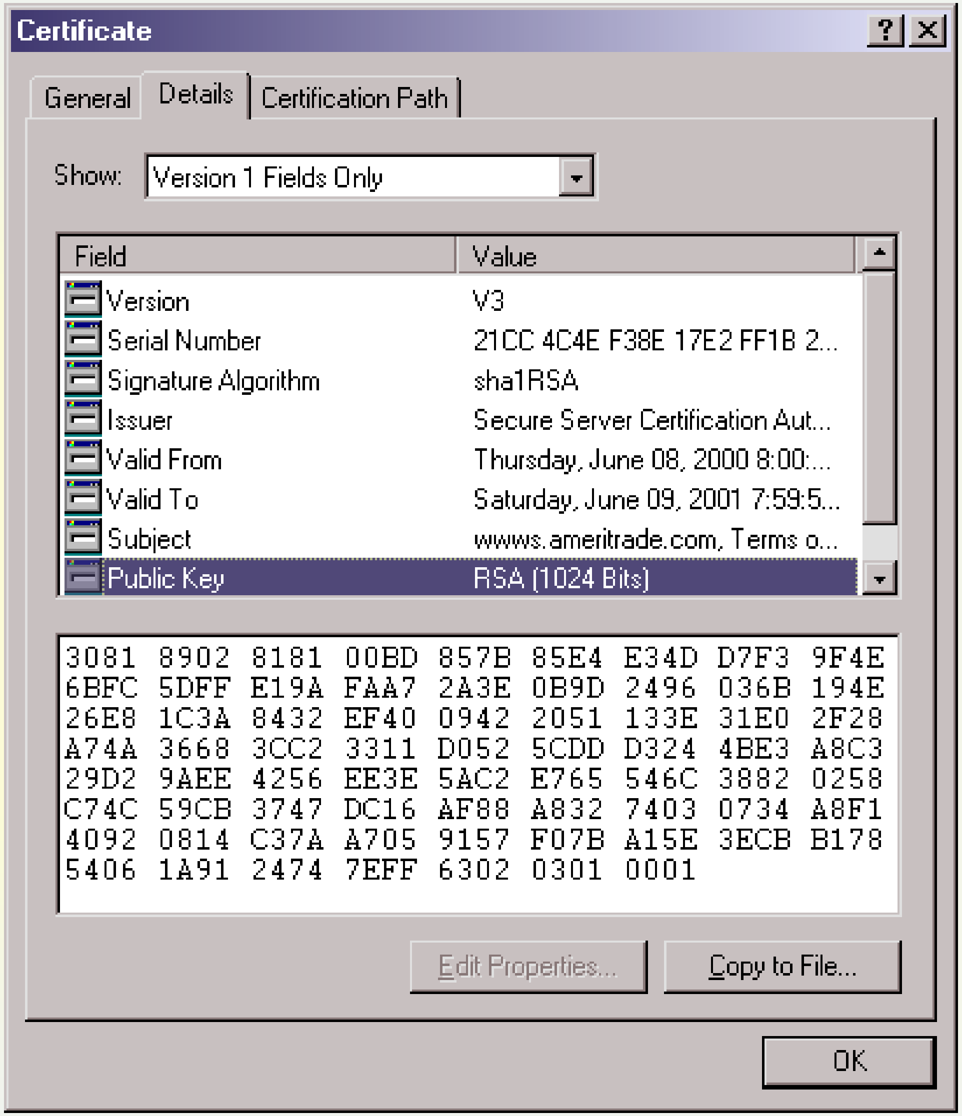
- ✓ Trusted third party like a government for passports
- ✓ CA authenticates that the public key belongs to Alice
- ✓ CA creates Alice's a Digital Certificate

Certificate



Data encrypted using secret key exchanged using some public key associated with some certificate.

Certificate



X.509 V3 Certificate Format

```
Certificate  ::=  SEQUENCE  {
     tbsCertificate            TBSCertificate,
     signatureAlgorithm        AlgorithmIdentifier,
     signatureValue            BIT STRING  }

 TBSCertificate  ::=  SEQUENCE  {
     version                   [0]  EXPLICIT Version DEFAULT v1,
     serialNumber               CertificateSerialNumber,
     signature                 AlgorithmIdentifier,
     issuer                    Name,
     validity                  Validity,
     subject                   Name,
     subjectPublicKeyInfo      SubjectPublicKeyInfo,
     issuerUniqueID            [1]  IMPLICIT UniqueIdentifier OPTIONAL,
                               -- If present, version shall be v2 or v3
     subjectUniqueID           [2]  IMPLICIT UniqueIdentifier OPTIONAL,
                               -- If present, version shall be v2 or v3
     extensions                [3]  EXPLICIT Extensions OPTIONAL
                               -- If present, version shall be v3
     }
```

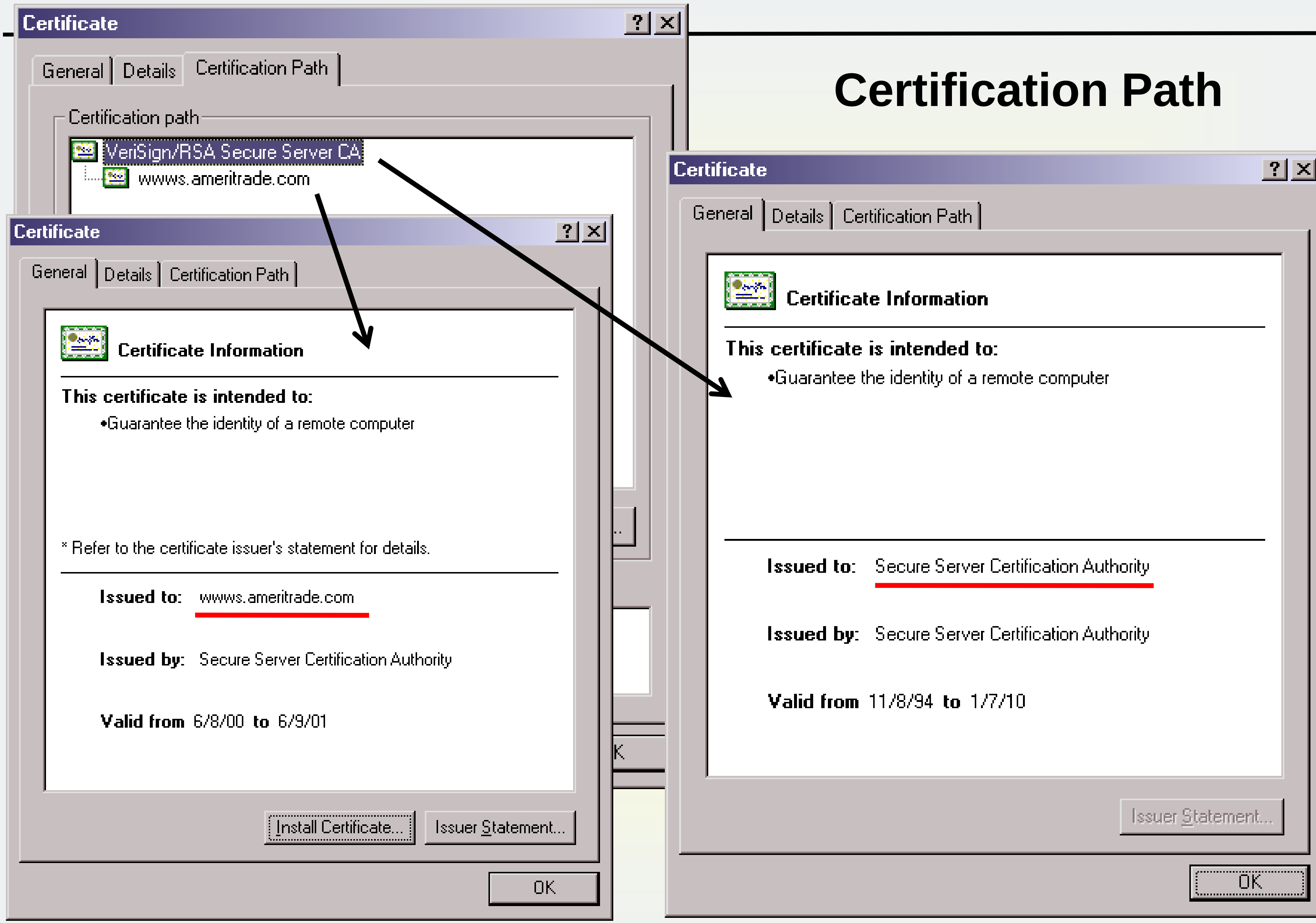
Sample Certificate

```
Certificate:
   Data:
      Version: v3 (0x2)
      Serial Number: 3 (0x3)
      Signature Algorithm: PKCS #1 MD5 With RSA Encryption
      Issuer: OU=Ace Certificate Authority, O=Ace Industry, C=US
      Validity:
         Not Before: Fri Oct 17 18:36:25 1997
         Not  After: Sun Oct 17 18:36:25 1999
      Subject: CN=Jane Doe, OU=Finance, O=Ace Industry, C=US
      Subject Public Key Info:
         Algorithm: PKCS #1 RSA Encryption
         Public Key:
            Modulus:
               00:ca:fa:79:98:8f:19:f8:d7:de:e4:49:80:48:e6:2a:2a:86:
               ed:27:40:4d:86:b3:05:c0:01:bb:50:15:c9:de:dc:85:19:22:
               43:7d:45:6d:71:4e:17:3d:f0:36:4b:5b:7f:a8:51:a3:a1:00:
               98:ce:7f:47:50:2c:93:36:7c:01:6e:cb:89:06:41:72:b5:e9:
               73:49:38:76:ef:b6:8f:ac:49:bb:63:0f:9b:ff:16:2a:e3:0e:
               9d:3b:af:ce:9a:3e:48:65:de:96:61:d5:0a:11:2a:a2:80:b0:
               7d:d8:99:cb:0c:99:34:c9:ab:25:06:a8:31:ad:8c:4b:aa:54:
               91:f4:15
            Public Exponent: 65537 (0x10001)
    Extensions:
         Identifier: Certificate Type
            Critical: no
            Certified Usage:
               SSL Client
         Identifier: Authority Key Identifier
            Critical: no
            Key Identifier:
               f2:f2:06:59:90:18:47:51:f5:89:33:5a:31:7a:e6:5c:fb:36:
               26:c9
```

```
   Signature:
      Algorithm: PKCS #1 MD5 With RSA Encryption
      Signature:
         6d:23:af:f3:d3:b6:7a:df:90:df:cd:7e:18:6c:01:69:8e:54:65:fc:06:
         30:43:34:d1:63:1f:06:7d:c3:40:a8:2a:82:c1:a4:83:2a:fb:2e:8f:fb:
         f0:6d:ff:75:a3:78:f7:52:47:46:62:97:1d:d9:c6:11:0a:02:a2:e0:cc:
         2a:75:6c:8b:b6:9b:87:00:7d:7c:84:76:79:ba:f8:b4:d2:62:58:c3:c5:
         b6:c1:43:ac:63:44:42:fd:af:c8:0f:2f:38:85:6d:d6:59:e8:41:42:a5:
         4a:e5:26:38:ff:32:78:a1:38:f1:ed:dc:0d:31:d1:b0:6d:67:e9:46:a8:
          dd:c4
```


-----BEGIN CERTIFICATE-----
MIICKzCCAZSgAwIBAgIBAzANBgkqhkiG9w0BAQQFADA3MQswCQYD
VQQGEwJVUzERMA8GA1UEChMITmV0c2NhcGUxFTATBgNVBAsTDF
N1cHJpeWEncyBDQTAeFw05NzEwMTgwMTM2MjVaFw05OTEwMTgw
MTM2MjVaMEgxCzAJBgNVBAYTAlVTMREwDwYDVQQKEwhOZXRzY
2FwZTENMAsGA1UECxMEUHViczEXMBUGA1UEAxMOU3Vwcml5YSB
TaGV0dHkwgZ8wDQYJKoZIhvcNAQEFBQADgY0AMIGJAoGBAMr6eZiP
GfjX3uRJgEjmKiqG7SdATYazBcABu1AVyd7chRkiQ31FbXFOGD3wNktb
f6hRo6EAmM5/R1AskzZ8AW7LiQZBcrXpc0k4du+2Q6xJu2MPm/8WKuM
OnTuvzpo+SGXelmHVChEqooCwfdiZywyZNMmrJgaoMa2MS6pUkfQVAg
MBAAGjNjA0MBEGCWCGSAGG+EIBAQQEAwIAgDAfBgNVHSMEGDAW
gBTy8gZZkBhHUfWJM1oxeuZc+zYmyTANBgkqhkiG9w0BAQQFAAOBgQ
BtI6/z07Z635DfzX4XbAFpjlRl/AYwQzTSYx8GfcNAqCqCwaSDKvsuj/vwbf
91o3j3UkdGYpcd2cYRCgKi4MwqdWyLtpuHAH18hHZ5uvi00mJYw8W2w
UOsY0RC/a/IDy84hW3WWehBUqVK5SY4/zJ4oTjx7dwNMdGwbWfpRqjd
1A==
-----END CERTIFICATE-----

Certification Path

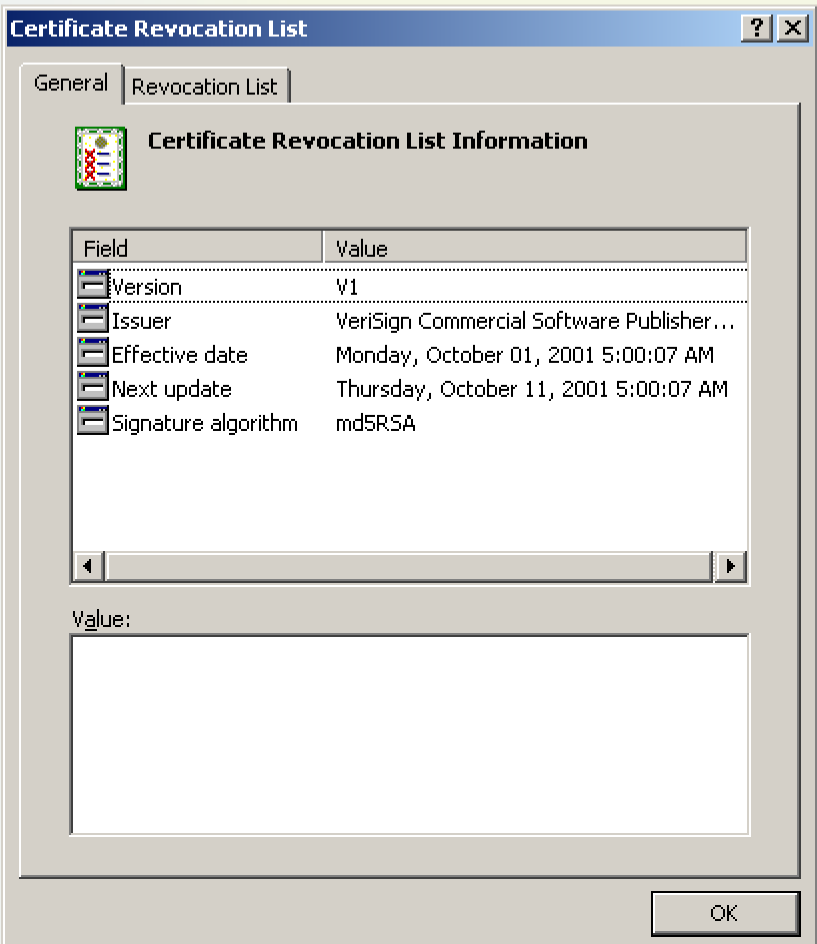


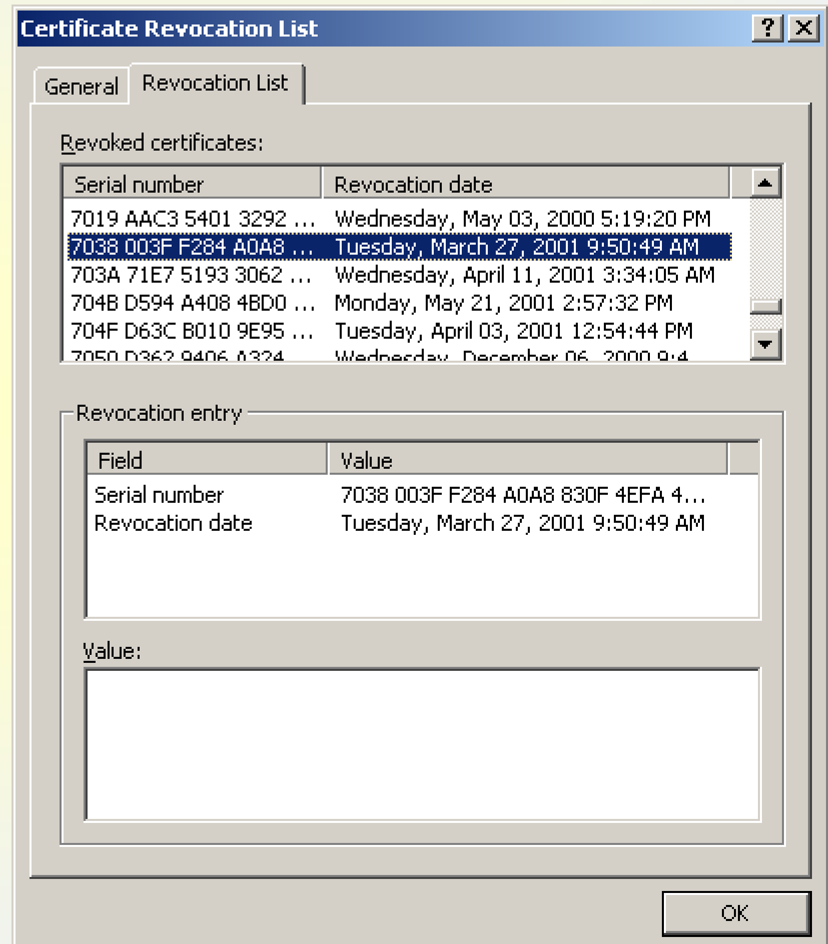
How to Revoke a Certificate?

- Certificate Revocation List (CRL)
  - A digital document which has a list of revoked certificates
  - Signed by CA
  - Defined in X.509 v2

- Why revoke a certificate?
  - When the user leave (retire from) the organization
  - Lost the private key, need to use a new key

Certificate Revocation List

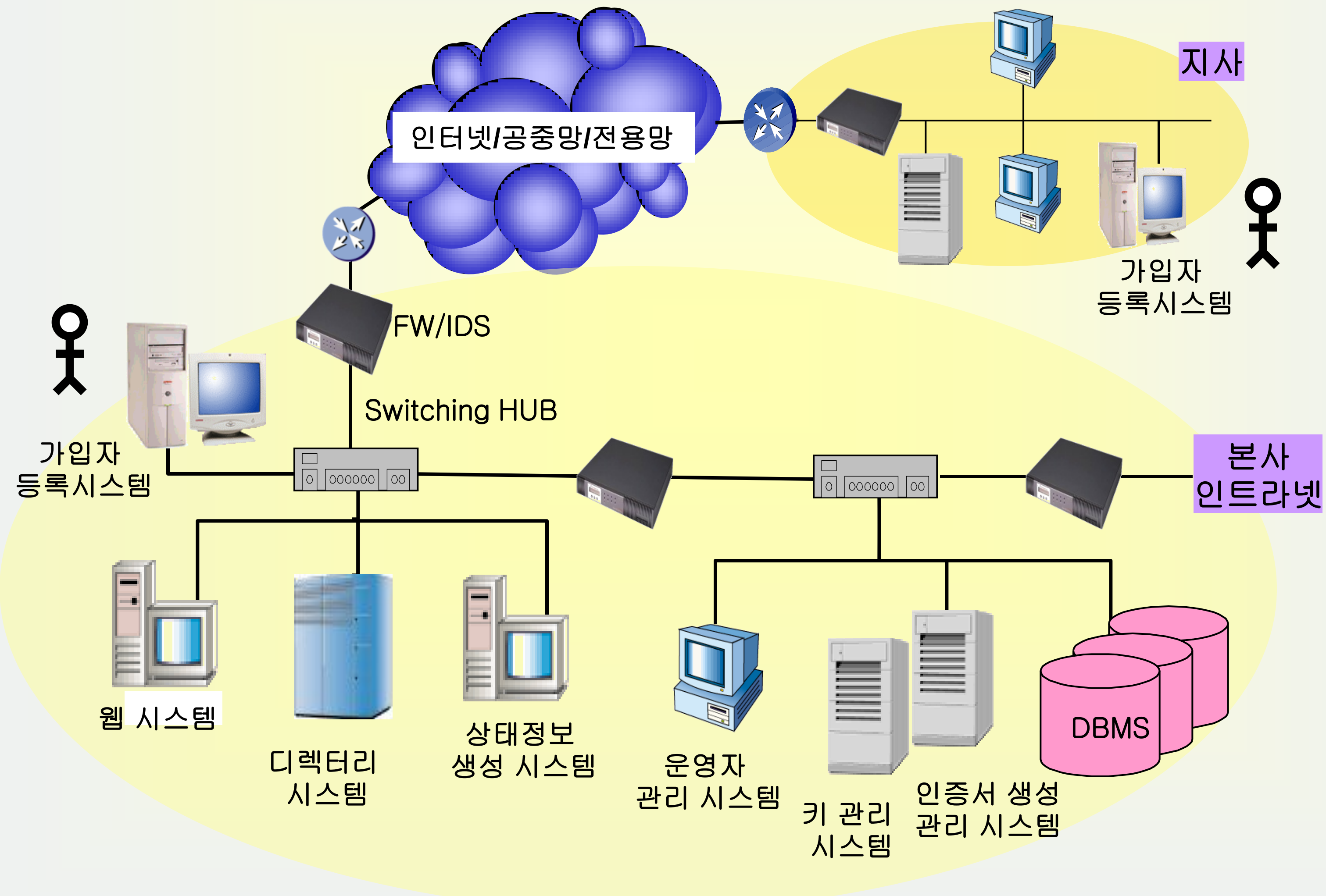




X.509 V2 Certificate Revocation List (CRL) Format

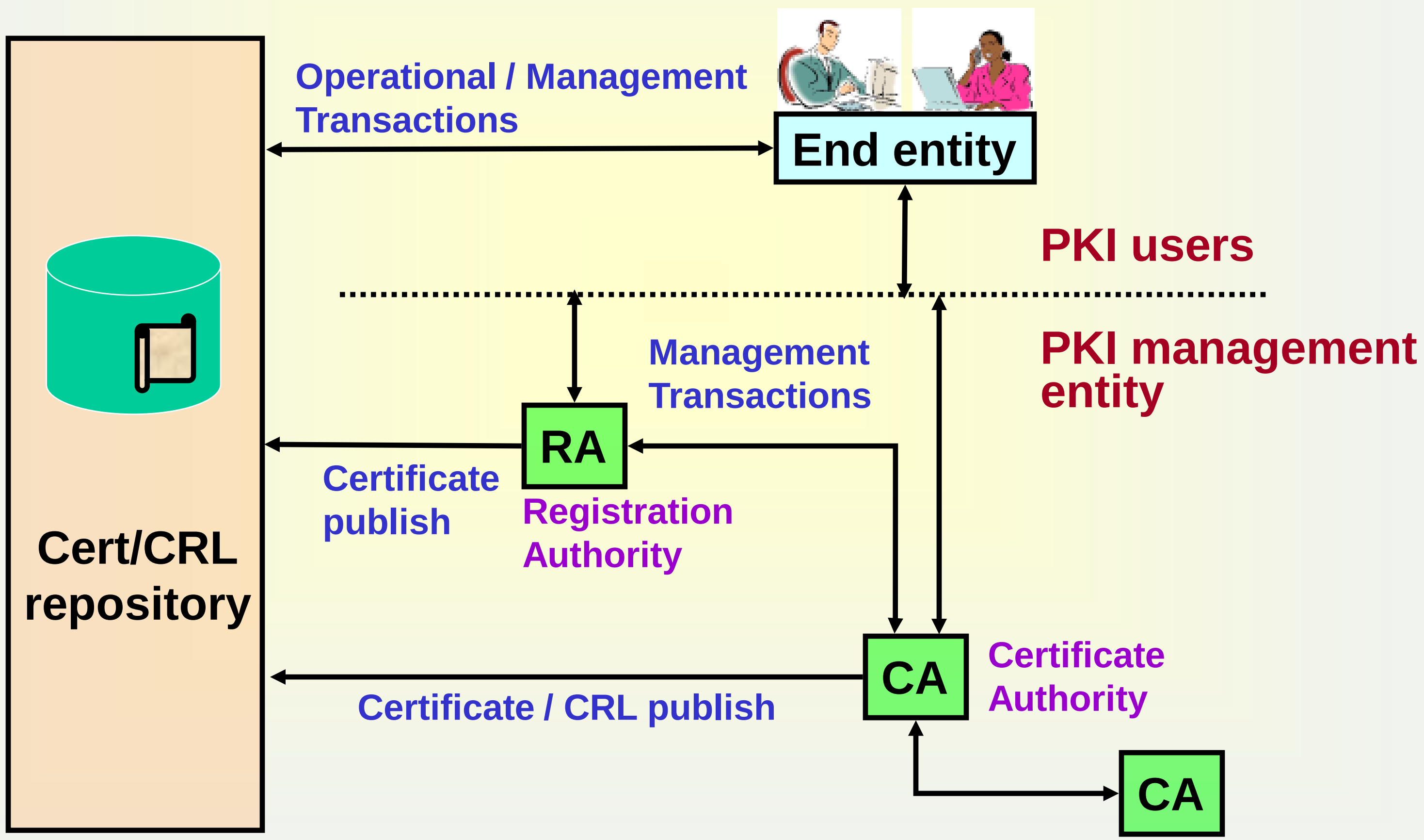
```
CertificateList ::=  SEQUENCE  {
     tbsCertList                TBSCertList,
     signatureAlgorithm         AlgorithmIdentifier,
     signatureValue             BIT STRING  }

 TBSCertList ::=  SEQUENCE  {
     version                    Version OPTIONAL,
                                 -- if present, shall be v2
     signature                  AlgorithmIdentifier,
     issuer                     Name,
     thisUpdate                 Time,
     nextUpdate                 Time OPTIONAL,
     revokedCertificates        SEQUENCE OF SEQUENCE  {
              userCertificate             CertificateSerialNumber,
              revocationDate              Time,
              crlEntryExtensions          Extensions OPTIONAL
                                          -- if present, shall be v2
              }  OPTIONAL,
     crlExtensions               [0]  EXPLICIT Extensions OPTIONAL
                                 -- if present, shall be v2
                       }
```

Overall Configuration of CA System

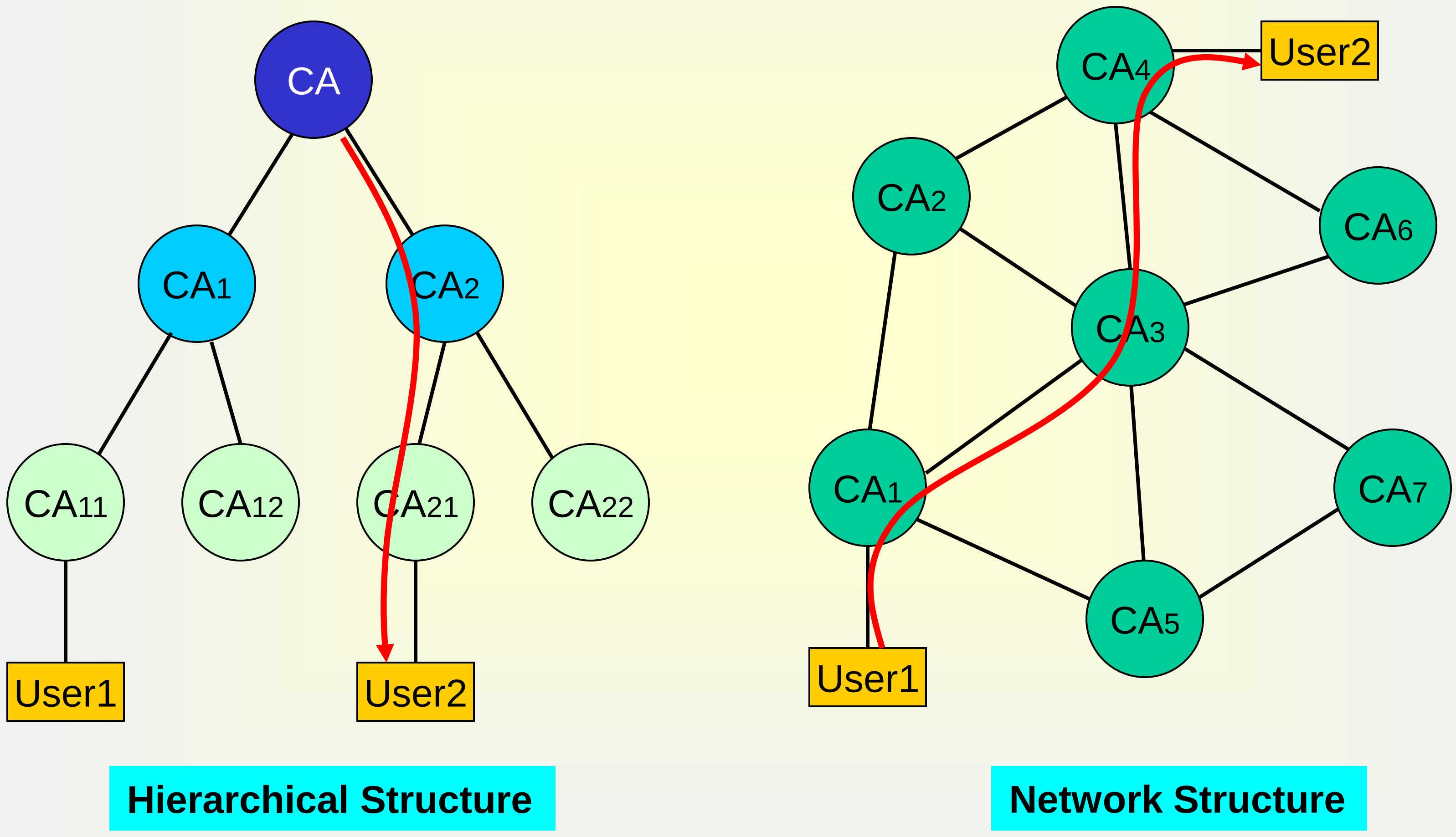


Public Key Infrastructure (PKI) Architecture

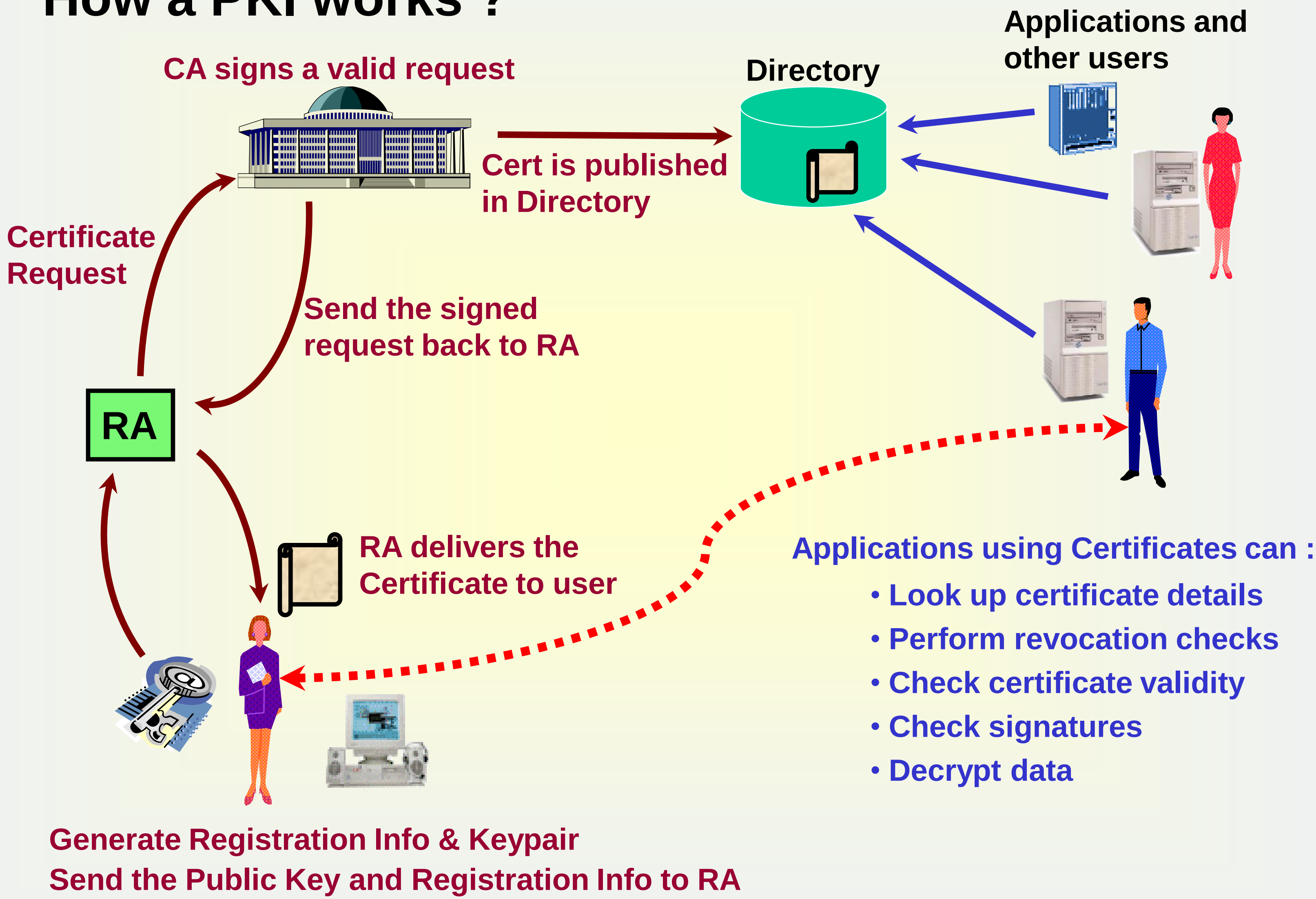
PKI is the hardware, software, people, policies, & procedures needed to create, manage, store, distribute, & revoke certificates



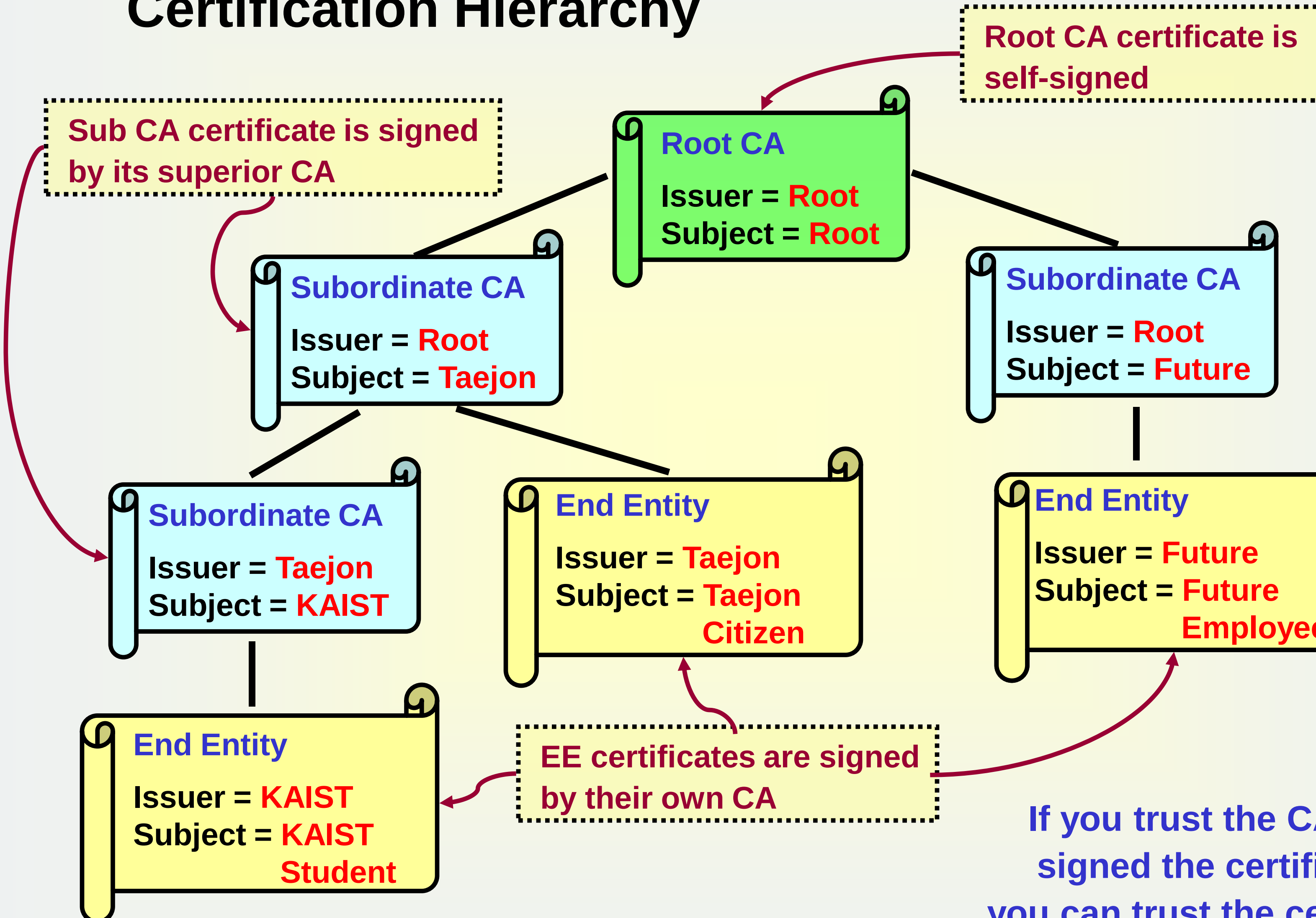
PKI Trust Relationship



How a PKI works ?



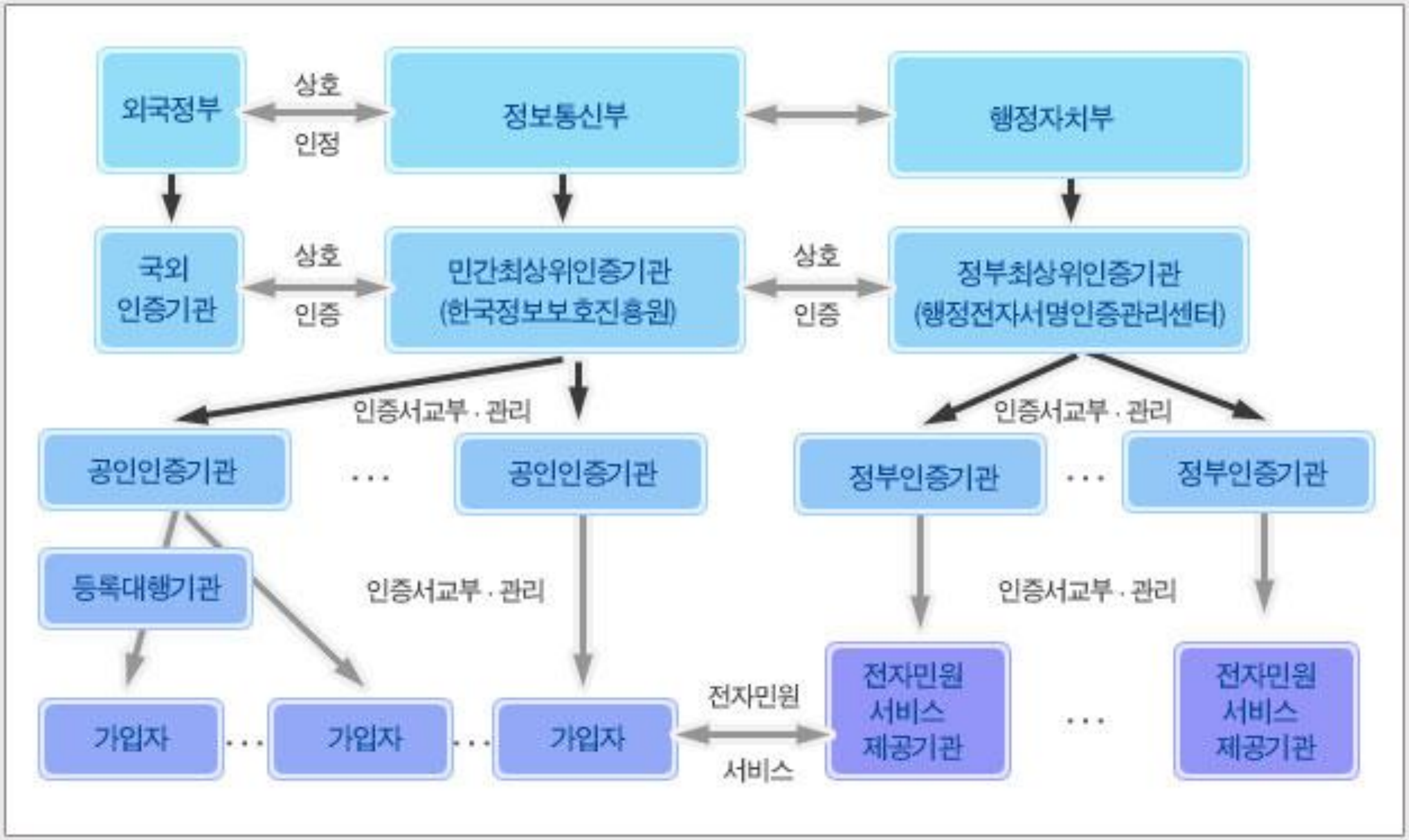
Certification Hierarchy



If you trust the CA that signed the certificate, you can trust the certificate

Korean PKI Structure

전자서명 인증관리센터
http://www.kisa.or.kr/kisa/kcac/jsp/kcac.jsp



Korean PKI Structure

전자서명법 제**4**조의 규정에 의하여 지정된 공인인증기관

- 한국정보인증**(**주**) http://www.signgate.com**
- **(**주**)**코스콤 **http://www.signkorea.com**
- 금융결제원 **http://www.yessign.or.kr**
- 한국정보사회진흥원 **http://sign.nca.or.kr**
- 한국전자인증**(**주**) http://gca.crosscert.com**
- 한국무역정보통신 **http://www.tradesign.net**

Homework #6

- Solve the exercises in this lecture

  Exercise 1: factorization using the quadratic sieve algorithm
  Exercise 2: Solve DLP using index calculus
  Exercise 3: RSA construction